UNIVERSIDAD DE DEUSTO

Facultad de Informática

DISEÑO DE UN MODULO DE OPTIMIZACION FISICA PARA ACCEDER A BASES DE DATOS EN RED, UTILIZANDO NUEVOS METODOS DE ACCESO

MEMORIA

QUE PARA OPTAR AL GRADO DE DOCTOR EN INFORMATICA

PRESENTA

María José Gil Larrea

DIRECTOR: Dr. D. F. Javier Zubillaga Zurimendi

MARZO, 1991

INDICE

RESUMEN

Para evaluar un query de usuario contra una base de datos en red, es necesario seguir los punteros o bien guardarlos para su posterior utilización. Hasta la fecha, el método utilizado para la evaluación de queries de red, ha consistido siempre en seguir los punteros, excepto cuando se trata de queries no arbóreos. Sin embargo, no siempre es necesario hacerlo, ni siempre se obtiene con dicho método un rendimiento mejor. Este trabajo de investigación presenta dos métodos originales de recuperación de datos, basados en el almacenamiento temporal de los punteros en relaciones intermedias. En dichos métodos se combinan los métodos de recuperación tradicionales, en red y relacionales, obteniéndose, en la evaluación de queries a bases de datos de red, un rendimiento sustancialmente mejor en la mayoría de los casos. Los métodos propuestos son útiles para la recupreción de información de bases de datos en red, así como de las relacionales implementadas con punteros. Para que los SGBD en red sean más asequibles, suelen presentar un interfaz que proporciona un lenguaje de interrogación de alto nivel, y un módulo optimizador que traduce los queries de usuario en queries de red eficientes. Nosotros presentamos los algoritmos de optimización por pasos. Primeramente se localiza un camino de acceso óptimo para un árbol de acceso dado, aplicando los métodos propuestos junto con el método tradicional, y a continuación, se presenta el algoritmo de enumeración de todos los árboles de acceso posibles. Para el proceso de queries complejos, que sólo pueden responderse uniendo los resultados de dos o más árboles de acceso, presentamos además un método heurístico.

ABSTRACT

To evaluate a query to network database it is necessary to either folow the links or save them. In previous research work the links have always been followed in order to answer the queries, except when they are not tree queries. However, it is not always nessary to do that, for a better performance is always obtain using this method. This research introduce two new methods for data retrieval, based on the temporal storage of the links in intermediate relations for later use. These methods, combine traditional network and relational retrieval methods, and show in most cases, a substancial improvement in performance in the evaluation of queries to network databases . The proposed methods are useful for retrieving data from network databases as well as from the relationals implemented with links. In order to make the network DBMS more user friendly, they usually have an interface with a high level query language and an optimicer module to translate the user queries into efficient network queries. In this research, we present the stepwise optimicer algoritm. First, an optimun access path is found for a given access tree using the proposed methods together with the tradicional one, and then, we apply an enumerating algoritm for all the access trees availables. We also introduce an heuristic method for procesing more complex queries which can only be answered by joining the results of two or more access trees.

RELACION DE FIGURAS

CAPITULO 1

INTRODUCCION Y OBJETIVOS

1.1 DESARROLLO HISTORICO

El inicio de los sistemas de gestión de base de datos (S.G.B.D.) se remonta a los años 60 y surge como una posible solución a los inconvenientes del entorno tradicional de explotación de la información, constituyendo hoy en día, una herramienta ampliamente aceptada, para reducir los problemas de gestión de grandes masas de información, de una forma eficiente.

Desde su aparición han evolucionado cualitativa y cuantitativamente, existiendo en nuestros días una gran diversidad de SGBD y de programas que pretenden serlo. Algunos de ellos, clasificados por el modelo de datos que soportan, son [SALT88]:

- Sistemas invertidos (Adabas, Datacom/DB, ...)
- Sistemas de ficheros encadenados (TOTAL, IMAGE, ...)
- Sistemas estrictamente jerárquicos (Delta, DEDB, ...)
- Sistemas bijerárquicos (IMS/VS, DL/I, ...)
- Sistemas Codasyl (IDMS, DBMS11, ...)
- Sistemas relacionales (Oracle, RDB, ...)

Generalmente, se suele hablar de tres modelos básicos:

- El Jerárquico, que engloba a los estrictamente jerárquicos y los bijerárquicos.
- El modelo en Red, que incluye la familia de ficheros encadenados y la de modelos Codasyl.
- El Relacional.

El modelo Jerárquico está caracterizado por sus estructuras de datos arborescentes, que son un conjunto ordenado de ocurrencias de un tipo de árbol. Con ellas implementa relaciones de grado 1:N (un padre tiene N hijos y cada hijo tiene un solo padre).

Los lenguajes de manipulación de datos en este modelo proporcionan un conjunto de operadores para el proceso de datos representados en forma de árboles; como por ejemplo, localizar un árbol concreto (la raíz es el punto de acceso), moverse de un árbol al siguiente, ..., etc.

En este modelo, también se incluye un soporte automático de ciertas formas de integridad referencial, pero plantea una serie de problemas:

- La falta de simetría (algoritmos correspondientes a preguntas simétricas tienen distinta complejidad).
- La escasa adaptación a las estructuras del mundo real.
- La redundancia de información.

Los SGBD en Red están presentes y seguirán estándolo en un extenso ámbito informático. Sus estructuras pueden verse como una 'extensión' de las del modelo Jerárquico, diferenciándose básicamente en que en el modelo en Red un hijo puede tener cualquier número de padres ,incluso cero. Con ellas se implementan relaciones N:M.

Las bases de datos están compuestas por ocurrencias de tipos de registros y de tipos de enlaces (sets). Un tipo de set relaciona un tipo de registro propietario con uno o más tipos de registros miembro. Cada ocurrencia de set consiste en una ocurrencia de registro propietario junto con un conjunto ordenado de ocurrencias miembro.

El uso de punteros, en estos sistemas, asegura el uso eficiente del espacio de almacenamiento, reduciendo los datos redundantes en la base de datos. Además, mediante los sets se soportan ciertas formas de integridad referencial.

El lenguaje de manejo de datos (D.M.L.), en el modelo en red, consta de un conjunto de operadores que permiten procesar datos representados en forma de registros y enlaces. A la base de datos se accede por registro (entidad) mediante un operador DML, por ejemplo dando el valor de algunos de sus campos. Para obtener la tupla deseada, se aplica secuencialmente una serie de operadores DML.

Esta forma de recorrido por la base de datos se denomina navegación. El propio usuario es el que debe indicar el camino concreto a seguir en la base de datos para acceder a la información deseada. Así, aunque los inconvenientes presentados por el modelo Jerárquico, como la falta de simetría y la inadaptación a las estructuras del mundo real, son resueltos en parte, el interfaz procedural para la manipulación de datos sigue sin liberar al usuario de las complejidades técnicas y sin proveerle de un acceso cómodo. En estos sistemas el rendimiento y la facilidad de uso son los objetivos principales.

En los años 70, con las proposiciones de Codd, aparecen los sistemas basados en **el modelo Relacional**, constituyendo una nueva generación de SGBD [CODD70].

La única estructura de datos que proporciona este modelo es la relación. Así, una base de datos es un conjunto finito de relaciones variables en el tiempo, definidas sobre un conjunto finito de dominios.

El lenguaje de manipulación está formado por un conjunto de operadores del álgebra relacional (o sus equivalentes del cálculo de predicados), para el manejo de datos en todas sus manifestaciones. Habitualmente, las relaciones se representan en forma de tablas, con lo que su manejo es sencillo.

La integridad de la información se garantiza en este modelo por medio de dos tipos de reglas:

- Integridad de entidades: Ningún atributo clave primaria de una tabla puede tener valores nulos o repetidos.
- Integridad referencial: A grandes rasgos consiste en que, si un atributo de una relación referencia al atributo clave de otra, todos los valores que aparecen en la primera deben figurar en la segunda.

Los SGBD deben lograr los objetivos de independencia de datos, no redundancia, capacidad de representación, integridad, seguridad y confidencialidad de los datos, accesibilidad y disponibilidad, de una forma eficiente; eficiente, en el sentido que minimicen el consumo tanto de recursos humanos como de recursos máquina, en el proceso de transacciones.

El coste de los recursos humanos, para la utilización de un SGBD, se mide en general por el alcance y la facilidad de uso de los lenguajes de definición y manejo que proporciona para cada tipo de usuario; y por el tiempo de respuesta. Los recursos máquina incluyen espacio de almacenamiento, tanto en memoria principal como auxiliar, y tiempo de U.C.P. y canales.

Lógicamente, el peso de cada uno de estos factores dependerá de la arquitectura del SGBD (si está geográficamente disperso o no) y del tipo de aplicaciones que se realizan. Por otro lado, aunque algunos de estos factores son complementarios, otros son opuestos; por lo que se hace necesario el establecimiento de ciertos límites para conseguir un grado de eficiencia aceptable.

1.2 PROBLEMATICA ACTUAL EN LA INTERACCION CON SGBD EN RED

El usuario interacciona con los SGBD expresando sus peticiones o 'queries', para selecionar y/o manipular datos. Para obtener un buen rendimiento, estas peticiones deben optimizarse.

En los sistemas de bases de datos en red, la optimización de queries se realiza habitualmente de forma manual; es decir, el propio usuario debe establecer el camino óptimo para recuperar la información deseada. Esto implica un conocimiento profundo de la organización física de las bases de datos, así como los caminos de acceso disponibles, datos estadísticos sobre la información almacenada en la base de datos, etc. Esta optimización también podría realizarla el propio SGBD, de forma automática, mediante un módulo optimizador.

Por otro lado, no cabe duda que la aproximación relacional representa la tendencia dominante en la oferta actual de sistemas de gestión de bases de datos. Sin embargo, en el mundo empresarial, es un hecho evidente, que las aplicaciones básicas y fundamentales se lleven a cabo en sistemas no relacionales, no pudiendo interrumpir

el proceso de mantenimiento que dichas aplicaciones conllevan, para rediseñarlas y modificarlas antes de poder utilizar un SGBD relacional.

En este escenario se hace necesario un SOFTWARE MIGRATORIO, en forma de un interfaz relacional. El mayor beneficio de este tipo de software es la eliminación de las costosas conversiones, ya que los programas existentes pueden seguir funcionando sin cambio alguno, y la disponibilidad inmediata de las facilidades del nuevo sistema.

Así, este interfaz proporciona a los usuarios un lenguaje de interrogación de alto nivel, o asercional, permitiéndoles definir los datos que desean obtener sin especificar el algoritmo de acceso. Esta característica permite acercar la base de datos a un número mayor de usuarios finales, que pueden formular preguntas, cuyo costo de procesamiento es elevado si se procesa directamente. Por esta razón, también en este caso, el sistema deberá realizar un tratamiento adicional que genere los algoritmos que permitan obtener una respuesta adecuada a las preguntas. Este paso lo realizará un módulo optimizador.

En cualquier caso, tanto si el sistema es relacional como si es en red y posea o no un interfaz relacional, la ventaja de este módulo no es sólo que el usuario

no tenga que preocuparse de cuál es la forma más eficiente de formular sus preguntas, sino que también ofrece la posibilidad de que seleccione una estrategia de procesamiento, aún mejor de la que pudiera seleccionar un programador. Esto se debe a que el módulo:

- puede tener acceso a información del tipo "valores en curso" que el programador no tiene
- puede examinar una mayor gama de alternativas que las contempladas por el programador.

En la mayoría de los sistemas relacionales, la optimización de preguntas se realiza en tres fases:

1.- Fase de análisis, consistente en el estudio sintáctico y semántico de la pregunta con el fin de verificar su corrección e incluso, simplificar el criterio de búsqueda.

2.- Fase de ordenación, consistente en descomponer la pregunta en una secuencia de operaciones elementales y determinar el órden óptimo de las mismas. La ordenación corresponderá a la generación de un plan de ejecución.

3.- **Fase de ejecución**, consistente en efectuar la secuencia de operaciones obtenidas en la etapa anterior con objeto de elaborar el resultado del query.

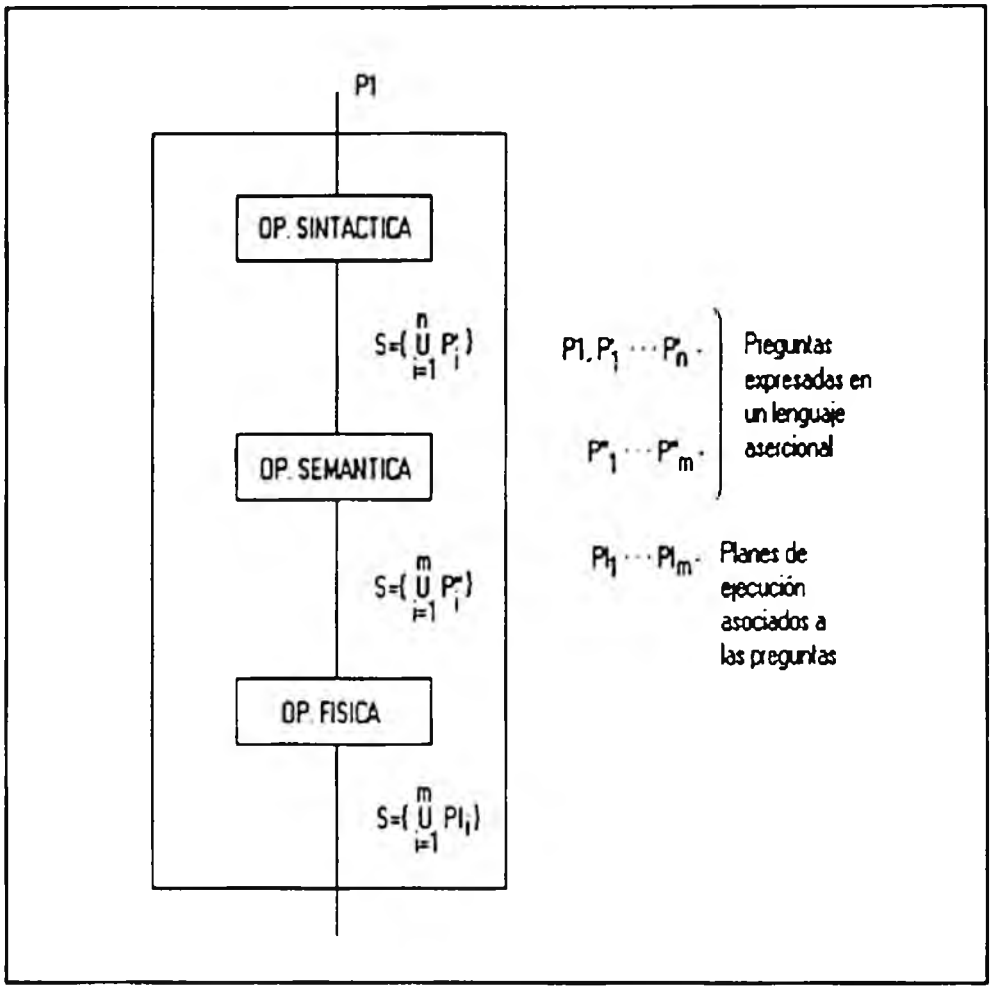


Figura 1.1

Arquitectura de un optimizador

1.3 OBJETIVO DE LA TESIS

El objetivo de esta investigación es diseñar un módulo de optimización física para sistemas de bases de datos en red, que puedan ser accedidos con un lenguaje tipo SQL.

El principal problema teórico al que nos enfrentamos es diseñar un sistema que permita seleccionar el plan de ejecución óptimo asociado a la pregunta formulada por el usuario, una vez optimizada sintáctica y semánticamente.

Las investigaciones realizadas hasta la fecha, al respecto [PETE81], [DAYA82], [KUCK82], [ILLA87], utilizan el método de recuperación de datos tradicional. Sin embargo, hemos comprobado que este método da lugar a recuperaciones redundantes, lo que influye negativamente en el rendimiento.

Por otro lado, habitualmente un query no procedural, puede responderse evaluando cualquiera de los diferentes árboles de acceso asociados a él y no uno como consideran estos autores. La complejidad del tiempo de cálculo para los algoritmos a aplicar a cada árbol es exponencial. Sin embargo, dado que el número de tipos de registros envueltos en un query es por lo general suficientemente pequeño, el tiempo para ejecutar cualquiera de estos algoritmos es aún factible.

En esta investigación mostramos un sistema decidible, que combina el método de recuperación por tuplas, con otros dos nuevos métodos.

En el capítulo tres, desarrollamos estos métodos y realizamos un estudio de sus rendimientos, utilizando la fórmula de coste de Dayal y Goodman, corregida.

Una vez comprobado que dichos métodos son factibles, en el capítulo cuatro, desarrollamos el conjunto de algoritmos que componen nuestro módulo de optimización física y que utilizan nuestros métodos en combinación con el tradicional, para encontrar el camino de acceso más económico en la evaluación de todos los árboles de acceso posibles.

CAPITULO 2

DEFINICIONES Y TERMINOLOGIA

Las definiciones particulares de nuestra investigación, las iremos dando a medida que las necesitemos. A continuación, presentamos las definiciones básicas del modelo relacional y del modelo en red que necesitamos.

2.1 MODELO RELACIONAL

El componente básico de una base de datos relacional es la relación. Cada fila de una relación se llama tupla de la relación. Las columnas se etiquetan con nombres de atributos, y sus valores provienen de los dominios a los que están asociados. En general denotamos por A,B,C,... a los atributos y por ...,X,Y,Z a los conjuntos de atributos. Un conjunto de atributos X={A,B,C} lo denotamos por ABC, y la unión de los conjuntos X e Y, por XY. La definición de los atributos de una relación, junto con el

nombre de la misma, forman el esquema de relación correspondiente (por un abuso negligente de la notación, se denota por R1 al conjunto de atributos de R1). Un esquema de base de datos relacional está formado por un conjunto de esquemas de relación $R_1, R_2, \ldots, R_n$, y una base de datos relacional es un conjunto finito de relaciones variables en el tiempo $r_1, r_2, \ldots, r_n$ correspondientes a los esquemas de relación dados.

Habitualmente, representamos las relaciones mediante tablas, en las que cada fila corresponde a una n-tupla. Estas representaciones deben cumplir las siguientes propiedades derivadas de la definición de relación:

- Cada fila debe contener una combinación de valores de atributos que permitan identificarla unívocamente.
- El orden de las filas no es significativo.
- El orden de las columnas, una vez asignado el nombre del atributo, no es significativo.

El ejemplo 2.1 nos muestra de manera gráfica los conceptos indicados.

Ejemplo 2.1

Sea el esquema de relación R1(NC,A,CP). El dominio de NC es el conjunto de todos los nombres de ciudades de Euskadi; el de A, es el conjunto de todos los códigos de dos caracteres, abreviaturas de provincias vascas; y el de CP, el conjunto de todos los códigos postales de siete dígitos. En la figura 2.1 se muestra una relación, r1, para el esquema de relación R1.

NC	A	CP
Agurain	A	E-01200
Arrasate	G	E-20500
Barakaldo	B	E-4890x
Bermeo	B	E-48370
Bilbo	B	E-480xx
Donibane-Garazi	BN	F-64220
Donibane-Lohizune	L	F-64500
Donostia	G	E-200xx
Eibar	G	E-20600
Gasteiz	A	E-010xx
Gernika	B	E-48300
Hondarribia	G	E-20280
Irun	G	E-20300
Iruñea	N	E-310xx
Lizara	N	E031200
Maule	Z	F-64130
Miarritze	L	F-64200
Sangotza	N	E-31400
Uztaritze	L	F-64480

Figura 2.1

r1, relación de ciudades de Euskadi

2.1.1 LENGUAJES DE INTERROGACION RELACIONALES

Los lenguajes de interrogación para el modelo relacional se clasifican, básicamente, en dos grupos:

1.- **Lenguajes algebráicos**, donde las preguntas se expresan aplicando operadores del álgebra relacional.
2.- **Lenguajes basados en el cálculo de predicados**, donde las preguntas describen el conjunto deseado de tuplas especificando el predicado que deben satisfacer.

Los operandos del álgebra relacional son esquemas de relación $R_1,\ldots,R_n$, y los operadores SELECT (σ), PROJECT (π), JOIN NATURAL (*), UNION (U), DIFERENCIA (-), etc. Una expresión del álgebra relacional se evalúa sustituyendo las relaciones $r_1,\ldots,r_n$, y aplicando los operadores según las definiciones. La clase de expresiones relacionales que se forma a partir de los operadores SELECT, PROJECT y JOIN es la de QUERIES SPJ.

El SQL es un lenguaje de interrogación intermedio entre el álgebra y el cálculo relacional. La operación fundamental en SQL está representada sintácticamente por la cláusula SELECT-FROM-WHERE. En términos algebráicos, una cláusula de este tipo podría considerarse como una selección (las expresiones lógicas aparecen en la

subcláusula WHERE), a partir (FROM) de las tuplas de la relación obtenida por una proyección sobre los atributos que aparecen en la subcláusula SELECT. Los queries complejos pueden componerse anidando cláusulas de este tipo.

Por otro lado, un interfaz basado en el concepto de relación universal, libera al usuario de tener que recordar la estructura de cada relación que compone la base de datos. El usuario interacciona con la base de datos como si estuviese compuesta por una única relación llamada relación universal. En este caso al 'pertenecer' todos los atributos a una misma relación, el formato de los queries será simplemente:

```
DISPLAY { [id-tupla.]nom-atr }, ...
        [ WHERE exp-log ]
```

2.1.2 PANELES

Un query puede representarse por diversas expresiones relacionales semánticamente equivalentes. Así, los paneles o 'tableaus' son un tipo de representación utilizado para simplificar queries, mientras que los grafos objeto y de operador se utilizan para detectar clases especiales de queries.

Un panel es una notación tabular, consistente en una o más filas, llamadas filas del panel; una fila resumen y un bloque de condiciones. Las columnas corresponden a los atributos del esquema de la base de datos. Las filas del panel contienen variables diferenciadas, denotadas por a_i, variables no diferenciadas, denotadas por b_i, y constantes de los dominios de los atributos. La fila resumen consiste en una serie de variables diferenciadas en una serie de columnas, y blancos en las otras. El bloque de condiciones recoge las condiciones sobre las variables del panel, que formula el usuario, y que son expresiones lógicas con constantes, variables, operadores aritméticos y operadores lógicos.

Ejemplo 2.2

Sea el esquema de relación del ejemplo 2.1. El query:

> "Obtener el nombre de todas las ciudades de Bizkaia (B) que tengan un código postal diferente al de Bilbao"

puede representarse por el panel mostrado en la figura 2.2.

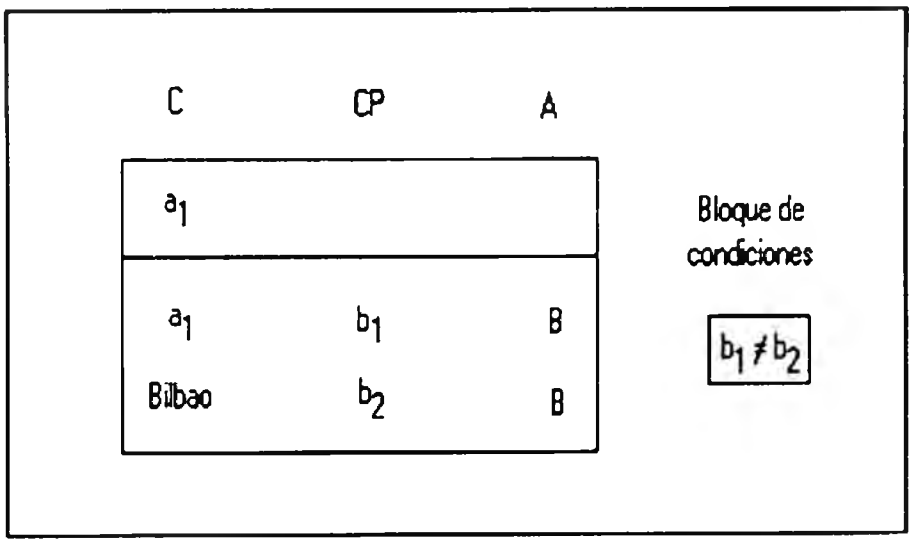


Figura 2.2

Panel del query del ejemplo 2.2

2.2 MODELO EN RED

Un esquema de una base de datos en red consta de una serie de tipos de registro, con cero o más campos, y sets que sirven para implementar relaciones 1:M entre registros tipo propietario y tipo miembro. El espacio total de almacenamiento de una base de datos en red se divide en áreas. Para cada tipo de registro, el esquema especifica el área o áreas en las que sus ocurrencias pueden ubicarse.

Para que esta terminología sea compatible con la relacional, llamaremos a los campos atributos (por un uso indebido de la notación, el conjunto de atributos de un tipo de registro R_i se denota por R_i).

Para cada tipo de registro se puede declarar una combinación de atributos, como clave de cálculo y, si el registro es de tipo miembro, cualquier combinación de atributos puede declararse clave de búsqueda. En la práctica, para cada set formado por un registro tipo propietario y un registro o más tipo miembro, pueden existir varias claves de búsqueda, pero sólo se permite una clave de cálculo por tipo de registro.

Las ocurrencias de un tipo de registro pueden agruparse en un orden específico (cronológicamente, clasificadas por algún criterio), de forma que las ocurrencias adyacentes quedan almacenadas físicamente lo más próximas posible. Un índice se conoce como índice de agrupamiento, si las ocurrencias del tipo de registro para el que se ha definido, se agrupan según el orden dado por dicho índice.

Un esquema de una base de datos en red, se puede representar mediante un grafo dirigido, donde los nudos corresponden a los tipos de registro, y cada arco une un tipo de registro propietario de un set con un tipo de registro miembro.

Los tipos de registro $R_1, R_2, \ldots, R_n$ forman un camino dirigido en el esquema, si se cumple la condición de que para todo i, $1 \leq i < n$, R_i es un tipo de registro propietario de R_{i+1} en algún set de dicho esquema (es decir, si hay un

enlace de R_i a R_{i-1}). Si hay un camino de R_i a R_j, entonces se dice que R_i es un ancestro de R_j, y R_j es un descendiente de R_i. También usamos los conceptos de ocurrencia propietaria, ancestro, miembro y descendiente. Supongamos que la ocurrencia de registro μ_i posee una ocurrencia de set, en la que μ_j es una ocurrencia miembro, entonces diremos que μ_i es la ocurrencia propietaria de μ_j, y μ_j es una ocurrencia miembro de μ_i. Si $\mu_1, \mu_2, \ldots, \mu_n$ son ocurrencias de registro, tales que para todo i, $1 \leq i < n$, μ_i es una ocurrencia propietaria de μ_{i-1}, entonces μ_1 es una ocurrencia ancestro de μ_n, y μ_n descendiente de μ_1. Al conjunto de todos los atributos del tipo de registro R_i y de todos sus ancestros, lo denotamos por R_i.

Ejemplo 2.3

Sea el esquema de la figura 2.3, que contiene los tipos de registro R1, R2, R3, y los sets S12, S32 y S31. R1 tiene una clave de cáculo ID_P, y los atributos NOM_P y APEL_P; R2, los atributos DNI y NOM_E, y una clave de búsqueda APEL_E; y R3 tiene una clave de cálculo COD_C. R3 es un tipo de registro propietario del set S31, del que R1 es un tipo de registro miembro. Los tipos de registros R3,R1,R2,

forman un camino dirigido en el esquema. R3 es un tipo de registro ancestro de R2, y R2 es un tipo de registro descendiente de R3.

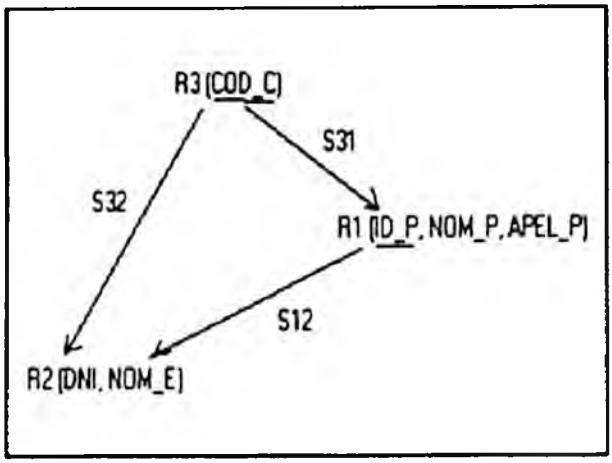


Figura 2.3

Grafo del esquema de red del ejemplo 2.3

2.2.1 QUERIES DE ARBOL

Un árbol de acceso T para un query sobre un esquema de red N, es un subgrafo de N, tal que la versión no dirigida de T es un árbol; es decir, que entre dos tipos de registros cualesquiera de T, haya exactamente un camino, y que todos los atributos que aparezcan en el query estén presentes en algún registro de T.

Un **camino de acceso** P de un árbol de acceso T, es una ordenación $R_{i1}, R_{i2}, \ldots, R_{in}$, de los registros de T, de forma que para todo j, $2 \leq j \leq n$, R_{ij} está conectado a uno de los tipos de registro $R_{i1}, \ldots, R_{ij-1}$ por un enlace del árbol de acceso.

Al evaluar un query, vamos pasando por los registros en el orden especificado por el camino de acceso. Cuando el orden de los registros no sea relevante para el método discutido, hablaremos sólo de árboles de acceso, asumiendo que un camino de acceso se crea y se usa antes de la evaluación de cada árbol. Un **query de árbol** contra una esquema de una base de datos en red N, es un query que puede responderse evaluando un árbol de acceso de N.

2.2.2 TRANSVERSALES

En el modelo en red, la forma básica de combinar registros correctamente, se define en términos de transversales. Sea $R_1, \ldots, R_n$ un camino del esquema, y μ_i una ocurrencia de R_i. $\mu_1, \ldots, \mu_n$, forman un transversal del camino, si para todo i, $1 < i \leq n$, μ_{i-1} es una ocurrencia propietaria de μ_i. Un transversal de camino completo es un transversal de camino $\mu_1, \ldots, \mu_n$ tal que R_1 no tiene otro propietario excepto SYSTEM.

Sea μ una ocurrencia del tipo de registro R. El r-transversal de R, se obtiene concatenando μ con todas sus ocurrencias ancestro; es decir, concatenando todos los transversales de camino completo que terminan en μ. La relación que se obtiene localizando todos los μ-transversales de cada ocurrencia μ de R, es el transversal de R o R-transversal.

2.2.3 CLAVES DE REGISTROS

El valor de la clave para una ocurrencia de registro μ de R, identifica únivocamente dicha ocurrencia en la base de datos. Esta clave puede ser física o lógica. Toda ocurrencia de registro tiene una clave de base de datos, que es su dirección física y única en la base de datos. Las claves lógicas vienen dadas por las claves de cálculo y las claves de búsqueda cuando están declaradas como únicas; es decir, no admiten valores duplicados.

Supongamos que $R_{j1},\ldots,R_{jm}$ es un camino de acceso en el esquema N, $X_1,\ldots,X_m$ son conjuntos no vacíos de atributos, tales que X_m es una clave de cálculo de R_{jm}, y para todo i, $1\leq i<m$, X_i es una clave de búsqueda para R_{j1} en el set poseído por R_{j1+1}, entonces $X=X_1\ldots X_m$ es una clave lógica de R_{j1} si para todo j, $1\leq j\leq m$, la clave X_j se declara como única.

2.2.4 IMPLEMENTACION DE SETS

Asumimos que todo set se implementa mediante una estructura de anillo doblemente encadenada, consistente en la ocurrencia propietaria y todas las ocurrencias miembro de una ocurrencia de set. Cada ocurrencia de registro apunta a la siguiente (puntero NEXT) y a la anterior (puntero PRIOR). Los punteros utilizados son las claves de base de datos de las ocurrencias a las que se apunta. Además de esta estructura, los sets pueden tener punteros OWNER, es decir, cada ocurrencia miembro apunta a su propietaria.

2.2.5 LENGUAJE DE MANIPULACION DE DATOS

Básicamente, hay cuatro formas de recuperar un registro de una base de datos en red, aunque para cada una de ellas, puede haber uno o más formatos [CODA78]. Cada formato tiene un comando FIND o FETCH. El comando FIND localiza una ocurrencia de registro de la base de datos y la establece como registro en curso de la transacción, actualizando apropiadamente el resto de indicadores de la tabla de registros en curso. El comando FETCH realiza la operación FIND y además recupera la ocurrencia.

1.- Si el registro que se va a recuperar es propietario de un set, y se va a recuperar como tal, el formato será:

FETCH/FIND OWNER WITHIN <nom-set>

2.- Si el registro se desea recuperar mediante su clave de cálculo:

FETCH/FIND RECORD <nom-reg> USING <clave-cálc.>

Si la clave de cálculo permite valores duplicados, el formato será:

FETCH/FIND NEXT DUPLICATE RECORD <nom-reg>
USING <clave-cálc.>

3.- Si el registro se va a recuperar como miembro de un set, el formato será:

FETCH/FIND | FIRST | <nom-reg> WITHIN <nomset>
| LAST |
| NEXT |
| PRIOR |

Si se dan valores constantes para uno o más atributos, entonces se inicializan dichos atributos y el formato será:

FETCH/FIND <nom-reg> WITHIN <nom-set>
USING <atr1,atr2,...>

o bien:

FETCH/FIND NEXT DUPLICATE WITHIN <nom-set>
USING <atr1,atr2,...>

4.- Si el registro se va a recuperar recorriendo secuencialmente todas las áreas que pueden contener ocurrencias de ese tipo, el formato será:

FETCH/FIND	FIRST LAST NEXT PRIOR	<nom-reg> WITHIN <nom-área>

2.3 EJEMPLO DE UNA BASE DE DATOS EN RED

A continuación describimos la base de datos en red que vamos a utilizar a lo largo de toda la investigación. La figura 2.4 muestra el esquema de esta base de datos, con los tipos de registro S (Suministrador), P (Pieza), C (Cliente), R (Representante de ventas) y O (Orden o pedido). SP es un registro auxiliar. Los registros S, P y C son miembros de los sets singulares ALL-S, ALL-P y ALL-C respectivamente.

Toda la base de datos está almacenada en un área (A1), y el tamaño de página utilizado es de 2048 bytes. Cada uno de los atributos S#, P#, COLOR, C#, R#, SALARIO, PRECIO_U y CANT, requiere para su almacenamiento, 4 bytes; y S_NOM, S_CIUDAD, P_NOM, C_NOM, C_CIUDAD y R_NOM requieren 20 bytes. En este ejemplo suponemos que todos los sets están declarados como MANDATORY AUTOMATIC; es

decir, para todo set S, cada ocurrencia de registro miembro de S tiene siempre un propietario en dicho set. Las claves de cálculo están formadas por los atributos subrayados. Junto con la descripción de cada tipo de registro se indica el numero de ocurrencias esperadas de ese tipo, en la base de datos.

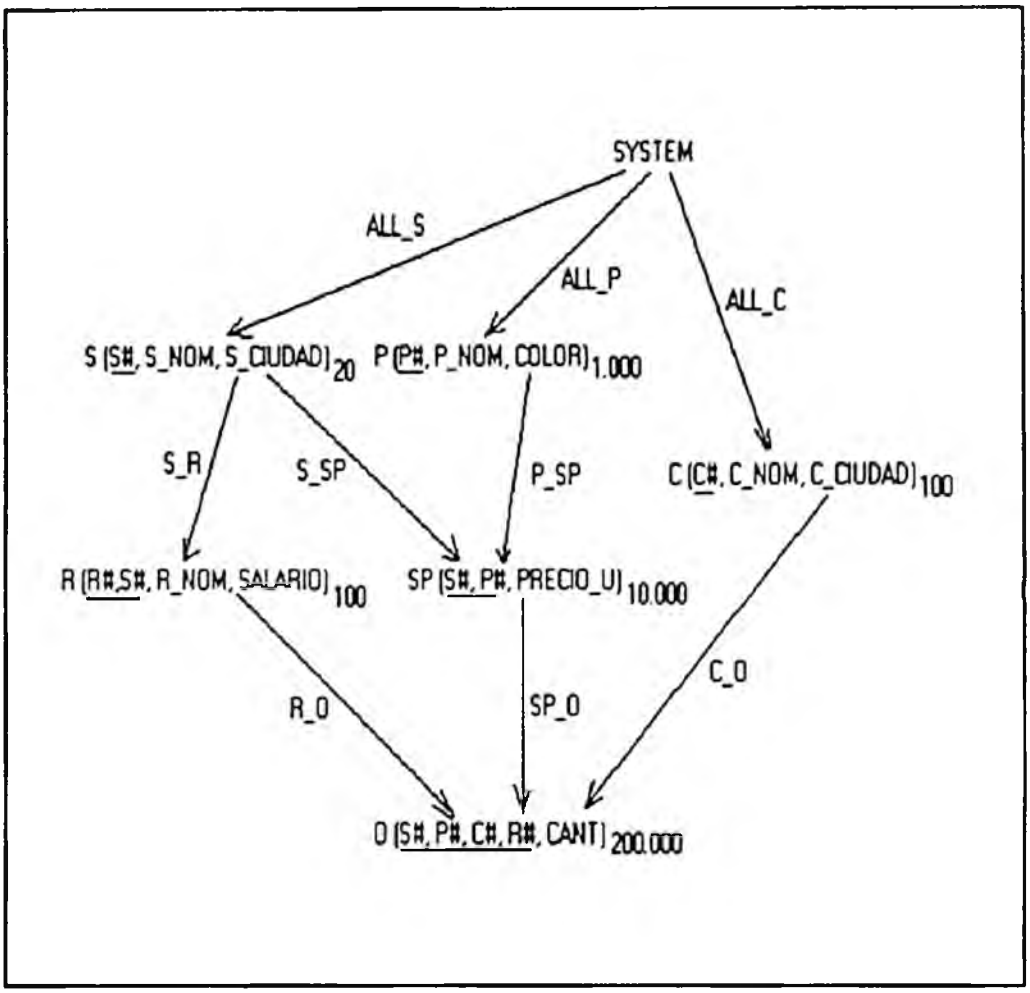


Figura 2.4

Esquema de la base de datos en red ejemplo

El esquema de la base datos relacional correspondiente, tiene seis esquemas de relación:

S(S#,S_NOM,S_CIUDAD)

P(P#,P_NOM,COLOR)

C(C#,C_NOM,C_CIUDAD)

SP(S#,P#,PRECIO_U)

R(R#,S#,R_NOM,SALARIO)

O(S#,P#,R#,C#,CANT).

CAPITULO 3

NUEVOS METODOS DE RECUPERACION

3.1 INTRODUCCION

La recuperación de información de una base de datos implementada con punteros se realiza, habitualmente, por tuplas; es decir, un operador DML recupera una ocurrencia de registro de una base de datos y para obtener una tupla resultado de la pregunta o query, se aplica en serie una secuencia de operadores DML. Para obtener la siguiente tupla resultado, empezamos nuevamente aplicando el primer operador de la secuencia y continuamos con el resto de operadores hasta que se haya aplicado el último. Por este procedimiento, se recupera la información siguiendo los punteros desde una ocurrencia de registro a la siguiente, hasta que todas las ocurrencias de registro de cada tupla requerida sean localizadas. Para que la conexión entre las ocurrencias de registro de una misma tupla no se pierda y sea imposible obtenerla, no tienen por qué seguirse siempre los punteros, sino que también pueden salvarse para su uso posterior.

Al estudiar investigaciones anteriores sobre el tema, hemos observado que no se ha considerado salvar los punteros, excepto para responder a los queries no arbóreos; es decir, queries evaluados mediante el acceso al mismo tipo de registro una y otra vez.

Por ejemplo, sea el esquema de red de la figura 2.4. El query:

```
DISPLAY t.S_NOM
WHERE s.S_NOM = "Fabrelec" AND t.P# = s.P#
```

requiere que se acceda al tipo de registro SP una vez por cada tupla del resultado, mientras 'se recuerda' la pieza en curso proporcionada por Fabrelec.

En esta investigación se demuestra que no siempre es necesario utilizar el método de recuperación por tuplas, para responder a un query de una base de datos implementada con punteros, ni siempre se obtiene con dicho método un número mínimo de accesos a páginas de disco. Para comprender ésto, examinemos la relación 1:N entre dos registros indicada por un puntero.

Sea un set S con un tipo de registro miembro R2 y un tipo de registro propietario R1. Muchas ocurrencias del registro R2 tienen la misma ocurrencia propietaria tipo R1. Para encontrar una tupla por el camino de acceso R2,R1, localizamos, primeramente, una ocurrencia de R2 y,

posteriormente, su ocurrencia propietaria de tipo R1. Para encontrar todas las tuplas resultado, por dicho camino de acceso, se repetirán estos pasos para cada ocurrencia de R2 en la base de datos.

De esta manera, estamos recuperando una ocurrencia de registro µ1 del tipo R1, no solo una vez, sino tantas veces como ocurrencias miembro haya en la ocurrencia del set, cuyo propietario es µ1. Por tanto, al incrementar el número medio de ocurrecias miembro en una ocurrencia de set, lo hace igualmente el número de accesos redundantes a páginas de disco.

En esta investigación, exponemos dos nuevos métodos para la evaluación de un query de esta naturaleza. Con ellos, una ocurrencia propietaria se recupera sólo una vez, independientemente del número de miembros de la ocurrencia de set que posee.

La aportación original de estos métodos respecto de los algoritmos de optimización desarrollados por otros investigadores [PETE81, DAYA82, KUCK82, ILLA87] es que en la evaluación de un query de red, no siempre seguimos los punteros, sino que los guardamos en una relación. Esta relación la combinamos, posteriormente, con los tipos de registro aún no recuperados, que están conectados a los punteros salvados.

Los métodos que desarrollamos, son útiles tanto para bases de datos en red, como relacionales que utilizan punteros, reduciéndose el tiempo de respuesta en un 50% para un query simple como el anterior. Esta reducción puede ser incluso mayor para queries más complejos.

Otros investigadores [CODD70, WONG76, SELI79, KIM82, DEWI84] no han considerado el uso de punteros para la implementación de las relaciones. Hoy en día, existen bastantes SGBD relacionales que los utilizan. Estudios comparativos de ambos tipos de SGBD, realizados recientemente, [GIL90], muestran que con la tecnología actual, las bases de datos implementadas con punteros tienen un mejor rendimiento.

Nuestros métodos combinan los métodos de recuperación de bases de datos en red y relacionales, y muestran cómo dividir un árbol de acceso en dos o más árboles disjuntos, donde los datos para cada uno se recuperan por tuplas, y se obtiene una relación de base de datos. Cada árbol, distinto al evaluado en principio, se extiende para incluir una de las relaciones obtenida de la evaluación de alguno de los otros árboles de acceso. Cada relación resultante es una relación de unión y contendrá una clave de algún tipo de registro del árbol de acceso al que ha sido añadido.

A continuación, exponemos el método tradicional y los nuevos métodos de recuperación de datos de una base de datos en red.

- Método de recuperación por tuplas.
- Método de clasificación de propietario.
- Método de división de árbol.

3.2 METODO DE RECUPERACION POR TUPLAS

Sea un árbol de acceso T, para un query de árbol Q, sobre un esquema de red N. Dado que cada camino de acceso $R_{i1},\ldots,R_{in}$ para dicho árbol, implica una ordenación de los tipos de registro a ser recuperados, únicamente habrá que generar el código necesario para recuperar R_{i1}, posteriormente R_{i2} y así sucesivamente hasta R_{in}. La recuperación de R_{i1} puede lograrse:

1.- Mediante una clave de cálculo, si se dan los valores de los atributos de dicha clave en el query.

2.- Mediante una clave de búsqueda, si R_{i1} es un tipo de registro miembro de un set singular, que tiene definida una clave de búsqueda y los valores para los atributos de dicha clave se dan en el query.

3.- Examinando secuencialmente todas las áreas que puedan contener una ocurrencia de registro de R_{i1}.

4.- Recuperando todas las ocurrencias de registro de tipo R_{i1}.

En este último caso R_{i1} puede recuperarse como un registro tipo miembro de un set singular, que contiene todas esas ocurrencias.

Si R_{i1} no es miembro de un set singular y $R_{j1},\ldots,R_{jk},R_{i1}$ es un camino directo de un esquema de red N, tal que R_{j1} es un tipo de registro miembro de un set singular, no apareciendo ninguno de los tipos de registro $R_{j1},\ldots,R_{jk}$ en T, entonces el camino de acceso puede extenderse y ser $R_{j1},\ldots,R_{jk},R_{i1},\ldots,R_{im}$. R_{j1} se recupera como registro tipo miembro de un set singular. El resto de los tipos de registro $R_{j2},\ldots,R_{jk}$ y $R_{i1},\ldots,R_{im}$ se recuperan bien como miembros o como propietarios de un set.

Supongamos que queremos recuperar R_{ij}. En el camino de acceso, R_{ij} está precedido por R_{ik} que es el único registro conectado a R_{ij} en el árbol de acceso por un puntero S. R_{ij} es o un tipo de registro propietario de S, o un tipo de registro miembro de S y se recupera según corresponda.

En la figura 3.1. se muestra una visión esquemática de un query para una base de datos en red. Cada línea horizontal denota el comienzo de un bucle en el que se recupera un tipo de registro del camino de acceso. Las flechas denotan el flujo de control cuando no existe la siguiente ocurrencia de registro. El código para imprimir la tupla recuperada está en el bucle más interno.

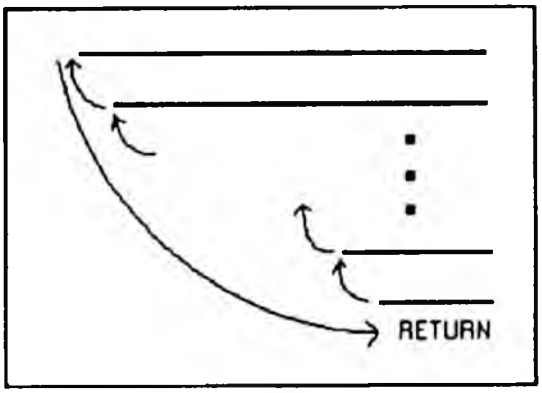


Figura 3.1

Visión esquemática de un query de red.

Después de recuperar cada registro, podría incluirse una comprobación, para aceptar o no la tupla, con una o mas expresiones de condición. Cada expresión se evalua tan pronto como es posible; es decir, inmediatamente después de que se recuperen los valores de todos los atributos que aparecen en la expresión.

Ejemplo 3.1

Sea el esquema de red de la figura 2.4 y el query siguiente:

"Para cada suministrador, listar todas las piezas que oferta".

```
DISPLAY S_NOM, P_NOM
```

A este query se puede responder correctamente siguiendo el camino de acceso S,SP,P.

Recuperando una tupla cada vez, se obtienen 20 ocurrencias del registro S, 10.000 del SP y 10.000 del P. Cada ocurrencia de registro P se recupera 10 veces.

A continuación, se muestra un query de red que se usa para implementar el camino de acceso S,SP,P.

```
        FETCH FIRST S WITHIN ALL-S
           ON 'set_vacío' GO TO 9999.
        GO TO 1500
1000    FETCH NEXT S WITHIN ALL-S
           ON 'fin_de_set' GO TO 9999.
1500       FIND FIRST SP WITHIN S-SP
              ON 'set_vacío' GO TO 1000.
           GO TO 2500.
2000       FIND NEXT SP WITHIN S-SP
              ON fin_de_set' GO TO 1000.
2500          FETCH OWNER P-SP CURRENT RECORD SP.
                  PRINT S_NOM, P_NOM
                  GO TO 2000
9999    RETURN
```

3.3 METODO DE CLASIFICACION DE PROPIETARIO

El método de clasificación de propietario es una estrategia para recuperar una sola vez una ocurrencia de registro como la propietaria de una ocurrencia de set, independientemente del número de ocurrencias miembro que tenga. Lógicamente, para recuperar un registro tipo R1 como un registro propietario de un set S, el tipo de registro R2 miembro de S debe preceder a R1 en el camino de acceso.

El método de clasificación de propietario recupera una parte del camino de acceso que incluye a R2, pero no a R1, y almacena el resultado en una relación temporal r (cuyo esquema de relación asociado se denota como R). Al mismo tiempo, se recupera un valor de clave para R1 y se almacena en r. Este valor de clave se obtiene sin un costo adicional si R2 contiene la clave de cálculo de R1, o si R2 tiene punteros 'owner' para el set S. En cualquier otro caso, obtenemos la clave de base de datos siguiendo los punteros de la estructura multilista.

Una vez obtenida r, la clasificamos por los valores de la clave del tipo de registro propietario R1; de ahí el nombre del método.

El camino de acceso que consideramos es R,R1. En vez de calcular una tupla de la forma tradicional, esto es, encontrando primero una ocurrencia de registro tipo R y recuperando la ocurrencia propietaria tipo R1 para cada tupla de r, sólo recuperamos una ocurrencia de registro R1 cuando el valor de la clave de R1 de la tupla en curso de R difiere del valor de la clave de R1 de la tupla anterior de R.

Comenzamos extrayendo del árbol de acceso T, el subárbol To, tal que los tipos de registro $Ro_1,\ldots,Ro_n$ de T, son registros de tipo propietario en algún set en el que el tipo de registro miembro está en To.

En el método de recuperación por tuplas, cada Ro_i ($1 \leq i \leq n$) se recupera como un registro tipo propietario (este tipo de registro es el que tenemos la intención de recuperar eficientemente). T, podría contener también tipos de registros $Rm_1,\ldots,Rm_p$ que son registros miembro en algún set donde el registro propietario está en To.

La figura 3.2 muestra To y todos los tipos de registro conectados a tipos de registro de To.

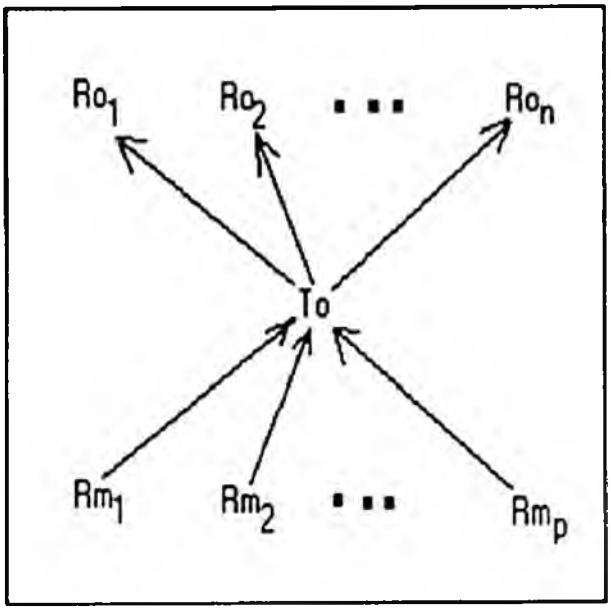


Figura 3.2

Tipos de registro de T conectados

a tipos de registro de To

Un árbol de acceso T, podría ser mucho más complejo que el mostrado en la figura 3.2. Si eliminamos los punteros de T que se muestran en esta figura tenemos:

- El subárbol To.
- Un árbol para cada Ro_i ($1 \leq i \leq n$).
- Un árbol para cada Rm_i ($1 \leq i \leq p$).

En la figura 3.3 representamos T como una conexión de estos subárboles.

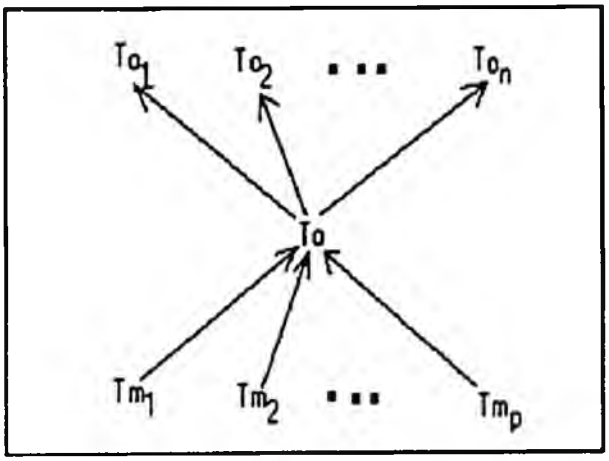


Figura 3.3

Arbol de acceso T representado como la conexión de n+p+1 subárboles.

El método de clasificación de propietario evalúa cada To_i ($1 \le i \le n$) mediante un árbol de acceso T_i consistente en To_i y un esquema de relación de unión o de 'join' $Rjoin_i$, como se muestra en la figura 3.4.

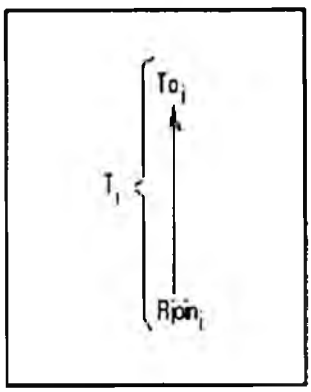


Figura 3.4

Arbol de acceso T_i para el método de clasificación de propietario.

La relación intermedia resultante de la evaluación de un árbol de acceso previo como To, o un árbol de acceso T_j ($j<i$) tiene a $Rjoin_i$ por esquema de relación.

Este esquema de relación contiene siempre una clave de registro del tipo Ro_i, y se clasifica siempre por el valor de esta clave antes de la evaluación de T_i. Otros atributos contenidos en $Rjoin_i$ son externos a esta evaluación, pero existen en $Rjoin_i$, a menos que se especifique lo contrario (a estos atributos los llamaremos atributos ajenos).

Sea la selección:

A <operación de comparación relacional> B

donde A está en algún tipo de registro en algún árbol ya evaluado, y B en un tipo de registro de algún árbol aún no evaluado.

Para una selección como ésta, el atributo A estará en $Rjoin_i$. También estará incluida en $Rjoin_i$, para todo árbol T_j aún no evaluado, la clave de Ro_j. Además debemos incluir una clave del registro propietario de cada Rm_j ($1 \leq j \leq p$).

El método de clasificación de propietario se puede aplicar utilizando una de las tres estrategias siguientes:

- Estrategia de una relación en serie
- Estrategia de dos relaciones en serie
- Estrategia paralela

A continuación mostramos su aplicación y su rendimiento, comprobando que la idoneidad de una estrategia depende del query en particular.

3.2.1 ESTRATEGIA DE UNA RELACION EN SERIE

Lo que caracteriza a esta estrategia es que, en cualquier momento, sólo hay una relación intermedia.

Primeramente, evaluamos To y obtenemos $Rjoin_1$. A continuación, evaluamos el árbol de acceso T_1 que contiene To_1 y $Rjoin_1$ obteniéndose $Rjoin_2$. En el momento i se evalúa el árbol de acceso T_i de la figura 3.4 y el resultado es $Rjoin_{i+1}$. Una vez que el árbol de acceso T_n se ha evaluado, se pasa a evaluar el árbol de acceso que contiene $Rjoin_{n-1}$, Tm_1, ..., Tm_p mostrado en la figura 3.5, completándose así el proceso del query.

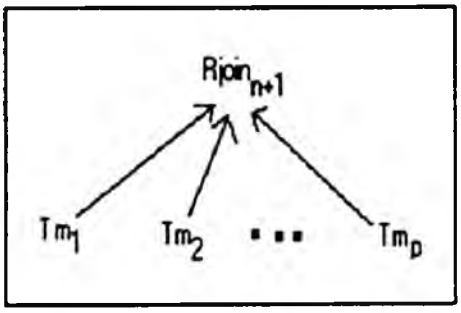


Figura 3.5

Ultimo árbol de acceso a evaluar.

El algoritmo para evaluar un query es el siguiente:

for cada árbol de acceso T_k ($0 \leq k < n$) **do**
  EVALUAR T_k y **GENERAR** $rjoin_{k-1}$;
  CLASIFICAR $rjoin_{k-1}$ por la clave de Ro_{k-1};
endfor;
EVALUAR T_n y **GENERAR** $rjoin_{n-1}$;
EVALUAR $Rjoin_{n-1}, Tm_1, \ldots, Tm_p$;

Puede observarse que no es esencial que el árbol de acceso que contiene los registros tipo miembro se evalue al final. En la mayoría de los casos, recuperar un registro como miembro de un set hará que el número de tuplas del resultado aumente. Las únicas excepciones se dan cuando la relación entre propietario y miembro no es 1:M sino 1:1, o cuando el tipo de registro miembro tiene

una probabilidad de selección muy baja para el query dado. Por esta razón, defendemos que el árbol de la figura 3.5 se evalúe el último, en todos los métodos. En cualquier caso, los tipos de registro conectados como miembro a un tipo de registro de To, tienen que justificarse, no siendo relevante esto último en los algoritmos que presentamos.

Ejemplo 3.2

Sea el query del ejemplo 3.1 y el camino de acceso S,SP,P usado para responder al query. Cuando se ejecuta con el método de clasificación de propietario, aplicando la estrategia de una relación en serie, el árbol de acceso To contiene los tipos de registro S,SP, y como resultado se obtiene $rjoin_1$ sobre $Rjoin_1$(S_NOM,P#). La relación $rjoin_1$ tiene 10.000 tuplas, necesitando cada una 24 bytes de memoria. La relación $rjoin_1$ se clasifica a continuación por P#. Una vez clasificada, se evalúa por tuplas el árbol de acceso T1 que contiene $Rjoin_1$,P. Cada página de $rjoin_1$ se recupera una vez y cada ocurrencia de P se recupera también sola vez.

3.2.2 ESTRATEGIA DE DOS RELACIONES EN SERIE

Principalmente, el coste de la estrategia de una relación en serie, viene dado por el de clasificar la relación de join. Este costo puede reducirse no incluyendo atributos ajenos en dicha relación. La extensión en bytes de una tupla se reduce, y así en un bloque pueden fijarse más tuplas. Los atributos sobrantes se guardarán en otra relación que denominaremos relación de mantenimiento. Una relación de mantenimiento contiene un resultado intermedio y debe combinarse con otra antes de ser unida con el árbol de acceso remanente, o bien, antes que se complete el proceso del query.

El esquema de la relación de mantenimiento lo denotamos por Rmant. La siguiente estrategia que se presenta utiliza relaciones de mantenimiento.

La estrategia de dos relaciones en serie es similar a la de una relación en serie, en la que los árboles de acceso se evalúan en la misma secuencia. Es decir, To se evalúa primero y después se evalúa uno a uno, los n árboles de acceso de una secuencia empezando por T_1 y así hasta T_n.

La estrategia de dos relaciones en serie se caracteriza por el hecho de que en cualquier momento hay dos relaciones de resultados intermedios. El resultado de

evaluar To es $Rjoin_1$ y $Rmant_1$, siendo $Rjoin_1$ y $Rmant_1$ relaciones paralelas. Es decir, la tupla i de $Rjoin_1$ y la tupla i de $Rmant_1$ es la tupla i del resultado obtenido al evaluar To.

Debemos incluir en la relación correspondiente al esquema de relación $Rjoin_i$ una columna conteniendo el número de tupla, o mejor un índice. Llamamos a esta columna INDICE de $Rjoin_i$. El atributo INDICE nos permite unir el resultado de evaluar T_i con $Rmant_i$. Para evaluar T_i, $Rjoin_i$ se clasifica por la clave de Ro_i. El resultado de evaluar T_i es Rmant y el valor de INDICE aparece también en este esquema de relación. Rmant se clasifica por INDICE. Así, $Rmant_i$ y Rmant pueden combinarse por el valor de este atributo. INDICE está implícito en la relación $Rmant_i$ ya que el orden de las tuplas en $Rmant_i$ nunca cambia. Los resultados de esta combinación son $Rmant_{i+1}$ y $Rjoin_{i+1}$.

Algunos atributos ajenos deberían darse en la relación de join, en vez de en la relación de mantenimiento para un mejor rendimiento.

Sea la selección siguiente:

A <operador de comparación relacional> B

donde A se da en algún árbol ya evaluado y B en el siguiente a evaluar (T_i). La forma más eficiente de ejecutar esta selección, es hacerlo durante la evaluación de T_i, inmediatamente después de recuperar el tipo de registro que contiene B, o bien durante la combinación de Rmant (relación de mantenimiento resultante de la evaluación de T_i) con $Rmant_i$.

En el primer caso, A debe estar en $Rjoin_i$ y en el último caso, B en Rmant. La mejor opción a elegir dependerá de los tiempos esperados de clasificación de $Rjoin_i$ y Rmant con y sin los atributos en cuestión.

El algoritmo para evaluar el query es el siguiente:

EVALUAR To y **GENERAR** $rmant_1$ y $rjoin_1$;
for cada árbol de acceso T_k ($1 \leq k < n$) do
    CLASIFICAR $rjoin_k$ por la clave de Ro_k;
    EVALUAR T_k y generar Rmant;
    CLASIFICAR rmant por INDICE;
    COMBINAR rmant y $rmant_k$ usando INDICE;
    GENERAR $rmant_{k+1}$ y $rjoin_{k+1}$;
endfor;
CLASIFICAR $rjoin_n$ por la clave de Ro_n;
EVALUAR T_n y **GENERAR** rmant;
CLASIFICAR rmant por INDICE;

COMBINAR rmant y $rmant_n$ usando INDICE

GENERAR $rjoin_{n-1}$;

EVALUAR $Rjoin_{n-1}$, $Tm_1, \ldots, Tm_p$;

Ejemplo 3.3

Sea el query del ejemplo 3.1 y el camino de acceso que se usa para responder a dicho query: S,SP,P.

Cuando se ejecuta con el método de clasificación de propietario de dos relaciones en serie, el árbol de acceso To contiene los tipos de registro S y SP, y da como resultado $rmant_1$, sobre $Rmant_1$(S_NOM) y $rjoin_1$ sobre $Rjoin_1$(P#,INDICE). La relación $rjoin_1$ se clasifica por P#. El árbol de acceso T1, que contiene $Rjoin_1$,P, se evalúa por tuplas obteniéndose rmant sobre Rmant(P_NOM,INDICE). Después, se clasifica la relación rmant por el atributo INDICE. Finalmente, $rmant_1$ y rmant se combinan por INDICE para responder al query.

3.2.3 ESTRATEGIA PARALELA

La estrategia paralela es similar a la estrategia de dos relaciones en serie en que los esquemas de relación de join no contienen atributos ajenos. Empezamos evaluando To y almacenando el resultado en n+1 relaciones paralelas correspondientes a los esquemas de relación $Rmant_0$, $Rjoin_1$, ..., $Rjoin_n$. Cada esquema de relación de join $Rjoin_i$, $1 \le i \le n$, está en T_i junto con To_i, y contiene INDICE como atributo, al igual que en la estrategia de dos relaciones en serie. $Rmant_0$ contiene sólo los atributos que son ajenos a la evaluación de cualquier To_i, $1 \le i \le n$. Ya que tenemos n relaciones de join, los n árboles de acceso $T_1, \ldots, T_n$ pueden ser evaluados en paralelo, lo que resultaría útil en un ambiente de multiproceso. El resultado de T_i, $1 \le i \le n$, es $Rmant_i$. Cada $Rmant_i$ se clasifica por INDICE. Como paso final, $Rmant_0$, $Rmant_1$, ..., $Rmant_n$ se combinan.

La figura 3.6 contiene un resumen de las estrategias para el método de clasificación de propietario. Los árboles de acceso de cada paso son los mismos para cada estrategia. La diferencia está en los resultados intermedios y en el número de clasificaciones y combinaciones requeridas. Cada relación de join $rjoin_i$, correspondiente al esquema de relación $Rjoin_i$, se clasifica por la clave de Ro_i antes de que se evalue T_i, incluso aunque no se muestre en la tabla.

Ejemplo 3.4

Sea el esquema de red de la figura 2.4 y el query:

DISPLAY R_NOM, C_NOM, P#, PRECIO_U, CANT

"Para cada representante de ventas obtener el identificativo, el precio por unidad y la cantidad de todas las piezas vendidas por él a cada cliente".

El árbol de acceso contendrá los tipos de registro C, O, R y SP. Para evaluar este query por el método de clasificación de propietario en paralelo, dividimos el árbol de acceso en tres subárboles T0, T1 y T2. T0 contendrá C y O; T1, $Rjoin_1$ y SP; y T2, $Rjoin_2$ y R.

En primer lugar evaluamos T0 por tuplas y obtenemos $rmant_0$, $rjoin_1$ y $rjoin_2$ sobre $Rmant_0$(C_NOM,CANT), $Rjoin_1$(S#,P#,INDICE) y $Rjoin_2$(S#,R#,INDICE), respectivamente. Después clasificamos $rjoin_1$ por S# P# y $rjoin_2$ por S# R#. Evaluamos T1 y T2, obteniendo las relaciones $rmant_1$ sobre $Rmant_1$(INDICE,P#,PRECIO_U) y $rmant_2$ sobre $Rmant_2$(INDICE,R_NOM). Ambas relaciones $rmant_1$, $rmant_2$

se clasifican entonces por INDICE. Finalmente, $rmant_0$, $rmant_1$ y $rmant_2$ se combinan obteniéndose el resultado del query.

Arboles de acceso a ser evaluados en cada paso	Estrategias en serie		Orden de ejecución		Estrategia paralela
	Estrategia de una relación	Estrategia de dos relaciones			
T_0	Recuperar T_0; Se obtiene $Rjoin_1$;	Recuperar T_0; Se obtiene $Rjoin_1$ y $Rmant_1$;	0	0	Recuperar T_0; Se obtiene $Rmant_0$; $Rjoin_1$,...,$Rjoin_n$
T_1: To_1 ↑ $Rjoin_1$	Recuperar T_1; Se obtiene $Rjoin_2$;	Recuperar T_1; Se obtiene Rmant; Clasificar Rmant; Combinar Rmant y $Rmant_1$; Se obtiene $Rmant_2$ y $Rjoin_2$;	1	1	Recuperar T_1; Se obtiene $Rmant_1$; Clasificar $Rmant_1$;
⋮	⋮	⋮	⋮	1	⋮
T_{n-1}: To_{n-1} ↑ $Rjoin_{n-1}$	Recuperar T_{n-1}; Se obtiene $Rjoin_n$;	Recuperar T_{n-1}; Se obtiene Rmant; Clasificar Rmant; Combinar Rmant y $Rmant_{n-1}$; Se obtiene Rmant y $Rjoin_n$;	n-1	1	Recuperar T_{n-1}; Se obtiene $Rmant_{n-1}$; Clasificar $Rmant_{n-1}$;
T_n: To_n ↑ $Rjoin_n$	Recuperar T_n; Se obtiene $Rjoin_{n+1}$;	Recuperar T_n; Se obtiene Rmant Clasificar Rmant Combinar Rmant y $Rmant_n$; Se obtiene $Rjoin_{n+1}$	n	1	Recuperar T_n; Se obtiene $Rmant_n$; Clasificar $Rmant_n$;
				2	Combinar R; $Rmant_1$,...,$Rmant_n$; Se obtiene $Rjoin_{n+1}$;

Figura 3.6

Resumen del método de clasificación de propietario.

3.4 METODO DE DIVISION DE ARBOL

Siempre que un tipo de registro R tiene más de un tipo de registro propietario en un árbol de acceso, y el método de recuperación es por tuplas, es necesario recuperar como propietario de un set todo registro tipo propietario, excepto uno.

En el método de división de árbol, dividimos el árbol de acceso en varios árboles, de forma que en cada uno, R tenga como máximo un registro tipo propietario. Dado que cada árbol tiene sólo un registro propietario Ro, este se recupera, bien como miembro de un set, o mediante una clave de cálculo. Las ocurrencias de registro de tipo Ro se recuperan una sola vez y no muchas, ya que esto se produce cuando un registro se recupera como propietario de un set.

Conceptualmente, subdividimos el árbol de acceso T en tantos árboles como registros tipo propietario haya. Sea un árbol de acceso T tal que R tiene como registros tipo propietario $Ro_1,\ldots,Ro_n$ y $Rm_1,\ldots,Rm_n$ como miembros. La figura 3.7 muestra los enlaces que conectan R con sus propietarios y sus miembros.

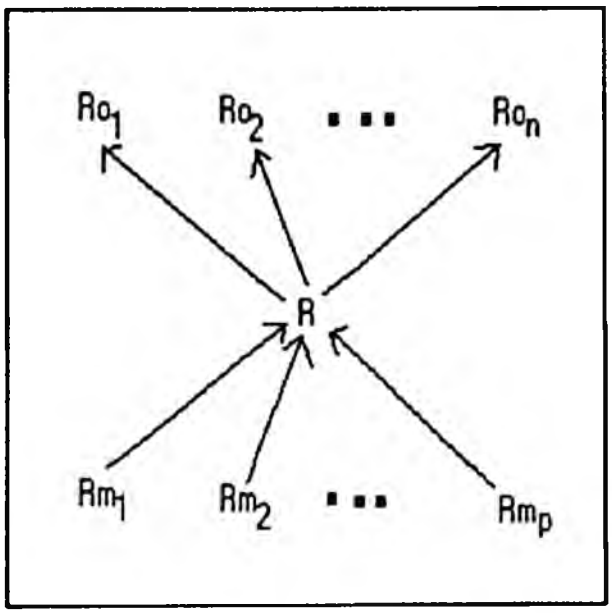


Figura 3.7

Tipos de registros conectados a R en T.

Si eliminamos de T el enlace entre R y Ro_k para todo k ($1 \leq k \leq n$), tendremos n+1 árboles; esto es, un árbol T_1 , ..., T_n por cada tipo de registro propietario $Ro_1, \ldots, Ro_n$, y un árbol T_{n+1} para la estructura del árbol de acceso.

La figura 3.8 muestra cómo se conectan entre sí los árboles de acceso que creamos a partir del árbol de acceso T. T_{n+1} podría estar conectado a cualquier árbol de acceso T_k ($1 \leq k \leq n$) ya que sólo es necesario evaluarlo una vez. Para que la evaluación sea eficiente, deberá depender de la posición de los atributos de la lista objeto y de los atributos de las selecciones.

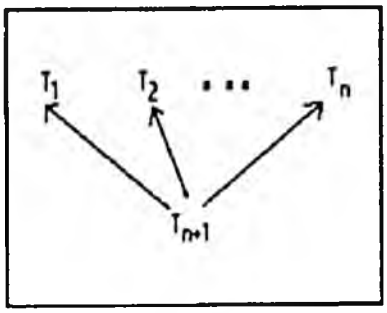


Figura 3.8

Conexiones entre los árboles creados.

Para cada uno de los n árboles de acceso, hay dos posibilidades. Sea el set S_k la conexión entre Ro_k y R. Si la estructura de datos utilizada para almacenar las ocurrencias del set S_k usa un índice denso, como un árbol binario equilibrado denso, en lugar de una estructura multilista simple, entonces no recuperamos R, sino sólo el valor de la clave de dicho índice. Denotamos esto en el árbol de acceso, por $\overline{R}$. La figura 3.9 resume el método de división de árbol.

Por cada árbol de acceso T_k ($1 \leq k \leq n+1$) hay un camino de acceso P_k. Los caminos de acceso $P_1, \ldots, P_n$ se evalúan en paralelo o en cualquier orden, antes de hacerlo con P_{n+1}. El esquema de relación $Rmant_k$ resultante de la evaluación de P_k ($1 \leq k \leq n$) contiene los atributos de la lista objeto, los de las selecciones, otros que no están en P_k, y una

clave de R. Una vez evaluados los n caminos de acceso, clasificamos cada $rmant_k$ resultante, para todo k, por la clave de R. A continuación, combinamos las relaciones clasificadas obteniéndose la relación $rmant_{n+1}$ sobre $Rmant_{n+1}$.

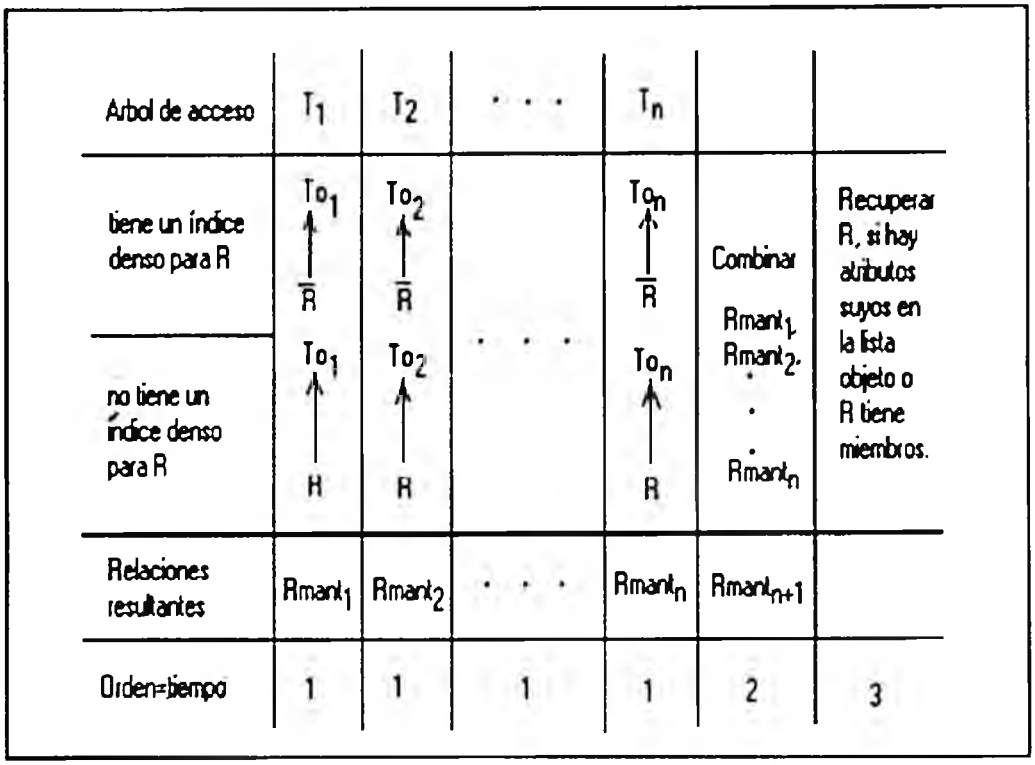


Figura 3.9

Resumen del método de división de árbol

Si todo set S_k, $1 \leq k \leq n$, tiene un índice denso, entonces R aún no ha sido recuperado. En este caso, añadimos $Rmant_{n+1}$ al camino de acceso P_{n+1}, como primer esquema de relación. La clave de R está en $Rmant_{n+1}$ y se utiliza para recuperarlo

eficientemente. Pero si al menos uno de los sets no tiene un índice denso, R ya se habrá recuperado. En este caso se reemplaza R por $Rmant_{n+1}$ en el camino de acceso P_{n+1}. El proceso del query se termina con la evaluación de este camino.

Ejemplo 3.5

Sea el esquema de red de la figura 2.4 y el query

```
DISPLAY S_NOM, P#, PRECIO_U
WHERE (S_CIUDAD = "Bilbao" OR
       S_CIUDAD = "Vitoria")
  AND P_NOM = "tornillo"
```

"Para cada suministrador de Bilbao o Vitoria, obtener el número de pieza y precio por unidad de los tornillos que proporciona".

Asumimos que: 5 de los 20 proveedores son de Bilbao o Vitoria; 400 de las 1000 piezas son tornillos; y tanto el set S-SP como P-SP tienen un índice denso.

El árbol de acceso T contiene los tipos de registro S, P, SP, y se subdivide en tres árboles que contienen S, P y SP respectivamente.

En primer lugar, recuperamos S y el índice denso para S-SP, y, a continuación, P y el índice denso para P-SP; obteniendo las relaciones resultado, $rmant_1$ y $rmant_2$ sobre $Rmant_1$(S_NOM,S#,P#), y $Rmant_2$(S#,P#), respectivamente. La relación $rmant_1$ tiene 2.500 tuplas, mientras que la relación $rmant_2$ tiene 4.000 tuplas. Después clasificamos $rmant_1$ y $rmant_2$ por S# y P#. Finalmente combinamos $rmant_1$ y $rmant_2$ usando S# y P#, y recuperamos SP, pero sólo aquellas ocurrencias cuyos valores de S# y P# aparecen en ambas relaciones $rmant_1$ y $rmant_2$.

3.5 MODELO DE COSTE

Usamos una versión revisada del algoritmo de coste dado por [DAYA82] para calcular el coste de un query. En primer lugar exponemos los parámetros de las estructuras de almacenamiento físico y la estimación del coste para cada paso de un query de red, y después, las fórmulas para el cálculo del coste esperado para dicho query.

3.5.1 PARAMETROS DEL MODELO DE COSTE

Para estimar el coste de un camino de acceso en la evaluación de un query contra una base de datos en red, necesitamos disponer de los siguientes parámetros:

a.- Para cada tipo de registro R.

nR: Número esperado de ocurrencias de registro.

Nr: Número esperado de páginas de disco, que contienen ocurrencias de R.

dI: Número esperado de accesos a páginas de disco, para acceder al índice I de R.

dH: Número esperado de accesos a páginas de disco, para acceder a R por su clave de cálculo.

dkey: Número esperado de páginas de disco que deben ser accedidas para encontrar todas las ocurrencias de R con un valor de clave de cálculo duplicado.

b.- Para cada área A:

dA: Número de páginas de disco en el área A.

c.- Para cada set S con un tipo de registro miembro M y un tipo de registro propietario O:

nM: Número esperado de ocurrencias miembro para una ocurrencia de set dada.

NM: Número esperado de páginas de disco que contienen una ocurrencia miembro para una ocurrencia de set dada.

σs: Probabilidad de que una ocurrencia de registro de tipo M sea una ocurrencia miembro de alguna ocurrencia del set S ($0 \leq \sigma s \leq 1$).

dI: Número esperado de accesos a disco para acceder al índice de las ocurrencias miembro de una ocurrencia propietaria (si el set S dispone de un índice I).

Además de estos parámetros se necesita también la probabilidad, σR, de que una ocurrencia de registro R satisfaga la condición de selección asociada.

3.5.2 ESTIMACION DEL COSTE

La evaluación de un query contra una base de datos en red, puede incluir las operaciones siguientes:

1.- Acceder a todas las ocurrencias de un tipo de registro que cumplan la condición de selección asociada.

2.- Acceder a todas las ocurrencias miembro que satisfagan la condición de selección asociada al propietario de una ocurrencia de set.

3.- Acceder al propietario de una ocurrencia de un set para una ocurrencia miembro dada.

4.- Evaluar un camino de acceso y guardar los resultados en una relación, clasificar la relación, y recuperarlos de nuevo para la evaluación de otro camino de acceso.

Para acceder a todas las ocurrencias de un tipo de registro R que cumplan la condición de selección asociada, el número esperado de accesos a páginas del disco dR es:

- si para acceder a R se usa la clave de cálculo

  dR=dH+dkey

- si la clave de cálculo se declara como DUPLICATES NOT

  dR=dH

- si se usa un índice de agrupamiento para acceder a R

  dR=Y(nR,NR,σR nR)+dI

donde Y(n, m, r) es el número esperado de páginas de disco de un fichero que tiene n ocurrencias de registro en m páginas de disco, que se acceden para recuperar del

fichero, r ocurrencias de registro. La fórmula exacta de Y la propuso Yao, y la aproximó posteriormente Berstein a:

$$Y(n, m, r) = \begin{cases} r & r \leq m/2 \\ (r+m)/3 & m/2 \leq r \leq 2m \\ m & 2m \leq r \end{cases}$$

1.- Si R está agrupado, pero no se usa ningún índice para acceder a R,

$$dR = NR$$

2.- Si para acceder a R se usa un índice que no es de agrupamiento,

$$dR = \sigma R \; nR + dI$$

3.- Si R es un tipo de registro miembro de un set singular y no puede usarse ninguno de los métodos anteriores,

$$dR = nR$$

4.- En cualquier otro caso, para todas las áreas A que contengan una ocurrencia de R,

$$dR = \sum_{A} dA$$

Para acceder a todas las ocurrencias tipo miembro M, que satisfacen la condición de selección para una ocurrencia propietaria O de un set S, el número esperado de accesos a páginas de disco dM es:

- si se usa un índice de agrupamiento

$$dM = Y(nM, NM, \sigma M\ nM) + dI$$

- si M está agrupado vía el set S, pero no se usa ningún índice

$$dM = NM$$

- si se usa un índice de no agrupamiento

$$dM = \sigma M\ nM + dI$$

- en cualquier otro caso.

$$dM = nM$$

Para acceder a la ocurrencia propietaria de tipo O el número de accesos a páginas de disco do es:

- si se dispone de punteros owner

$$do = \sigma s$$

- si no hay punteros owner y las ocurrencias miembro están agrupadas vía el set S

$$do = \sigma s/2\ (NM + 1)$$

- en cualquier otro caso

$$do = \sigma s/2\ (nM + 1)$$

Sea b el tamaño de página de disco; M páginas, la memoria disponible para buffers y t=M-1.

Para salvar o recuperar una relación con n tuplas de L bytes se necesitan p=(n/(b/L)) accesos a páginas de disco.

Para clasificar esa relación en un orden concreto se necesitan:

- $2p((\log_t (p/2t)) +1)$ accesos a páginas de disco si $p>2t$
- $2p$ si $p \leq 2t$.

3.5.3 CALCULO DEL COSTE DE UN QUERY PARA UNA BASE DE DATOS EN RED

La figura 3.1 muestra una visión esquemática de un query de red que se evalúa por el método tradicional (una tupla de cada vez). El número esperado de accesos a páginas de disco para ese query viene dado por el sumatorio de todos los bucles del query.

$$E = \sum_i (n_i \times E_i)$$

E_i es el número esperado de accesos a páginas de disco en cada ejecución del bucle i-esimo, y n_i es el número esperado de veces que el bucle i-esimo se ejecuta durante el proceso del query de red.

Ejemplo 3.6

Sea el query de red del ejemplo 3.1. El bucle que accede al tipo de registro S se ejecuta sólo una vez ya que es el bucle externo. Dado que se recuperan 20 ocurrencias de S, el número esperado de accesos a páginas de disco es 20. El bucle que accede al registro tipo SP miembro del set S-SP se ejecuta 20 veces; es decir, una por cada propietario de tipo S. El número esperado de ocurrencias miembro por cada ocurrencia de set es 500. Ya que las ocurrencias miembro no están agrupadas, el número esperado de accesos a páginas de disco en cada ejecución de este bucle es 500. Por tanto, el coste esperado para acceder al tipo de registro SP es de 20x500=10.000 accesos. El bucle que accede al tipo de registro P, propietario del set P-SP se ejecuta 10.000 veces. El número esperado de accesos a páginas de disco en cada ejecución de este bucle es 1 (el set P-SP está declarado MANDATORY AUTOMATIC) y el coste esperado para acceder al tipo de registro P es de 10.000

accesos a disco. Por tanto, el coste esperado para acceder al tipo de registro P es de 10.000 accesos a disco, y para responder al query 20020.

Cuando este query se ejecuta usando el método de clasificación de propietario con una relación en serie, el primer árbol accedido contiene registros S y SP, y se obtiene $rjoin_1$ sobre $Rjoin_1$(S_NOM,P#). Cada tupla de $rjoin_1$ requiere 24 bytes de memoria. Para salvar $rjoin_1$ que contiene 10.000 tuplas, se necesitan 118 accesos a páginas de disco. La relación $rjoin_1$ se clasifica por P# con un coste aproximado de 708 accesos a disco (suponemos que tenemos un buffer disponible de 5 páginas).

A continuación, se evalúa por tuplas el segundo árbol de acceso, el cual contiene $Rjoin_1$ y P. Cada página de $rjoin_1$ se recupera una vez y cada ocurrencia de registro de P se recupera también una vez por cada 1.118 accesos a disco. Por tanto, el tiempo para recuperar P es de 1944 accesos. El coste de ejecución del query con este método es de 10.000+20+1944=11.964 accesos a páginas de disco, aproximadamente un 40% menos que con el método tradicional.

3.6 COMPARACION DE RENDIMIENTOS

La figura 3.9 muestra el rendimiento del método de clasificación de propietario con una relación en serie frente al método de recuperación por tuplas, para el caso en que haya un total de 100.000 ocurrencias de registros miembro y cada ocurrencia de set tenga 1, 2, 4 o 16 ocurrencias miembro (el número de ocurrencias propietario es 100.000, 50.000, 25.000, 6250 respectivamente).

Esta figura se compone de cinco gráficas:

[1] Coste para el método de recuperación por tuplas.

[2] Incremento de velocidad para el método de clasificación de propietario con un miembro por ocurrencia de set.

[3] Incremento de velocidad para el método de clasificación de propietario con dos miembros por ocurrencia de set.

[4] Incremento de velocidad para el método de clasificación de propietario con cuatro miembros por ocurrencia de set.

[5] Incremento de velocidad para el método de clasificación de propietario con dieciseis miembros por ocurrencia de set.

El eje vertical es el incremento de la velocidad, que se define como el coste del método tradicional de recuperación de información, dividido por el coste del método que se compara. Es decir, si la velocidad es 1, entonces el método que se compara tiene el mismo coste que el método por tuplas y si la velocidad es 2, significa que el método que se compara es el doble de rápido (o tiene un rendimiento específico dos veces mayor) que el método tradicional.

El eje horizontal es el factor de bloqueo para la relación de join, indica cuántas tuplas de la relación de join pueden mantenerse en una página de disco.

Se asume que tenemos un buffer disponible de 5 páginas. El rendimiento del método de clasificación de propietario, con una relación en serie se ve afectado por 4 factores:

1.- El número de ocurrencias del tipo miembro.
2.- El número de ocurrencias miembro por ocurrencia de set.
3.- El número de páginas de memoria disponibles en el buffer.
4.- El número de tuplas de la relación de join que puede almacenarse en una página de disco, (es decir, el factor de bloqueo).

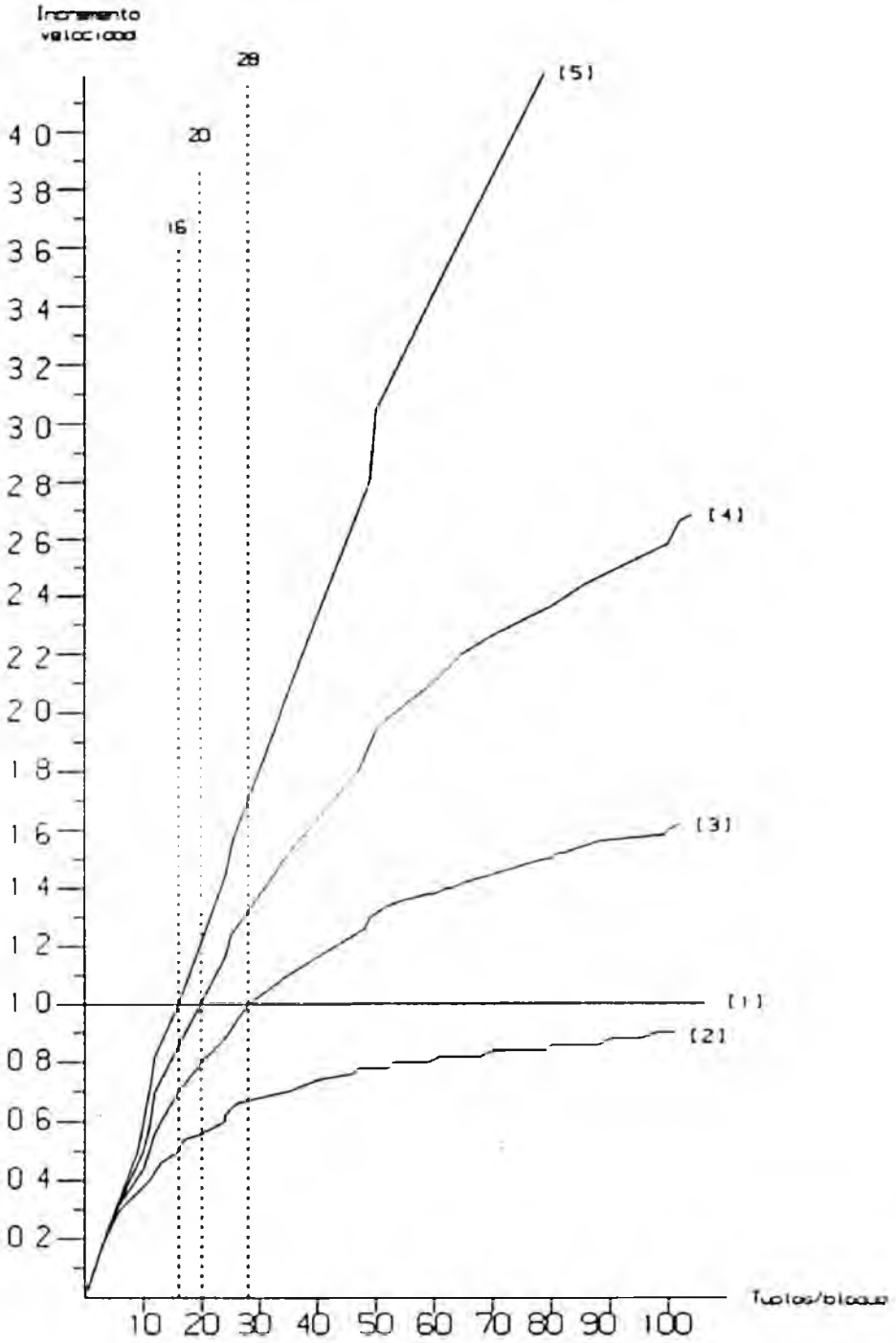


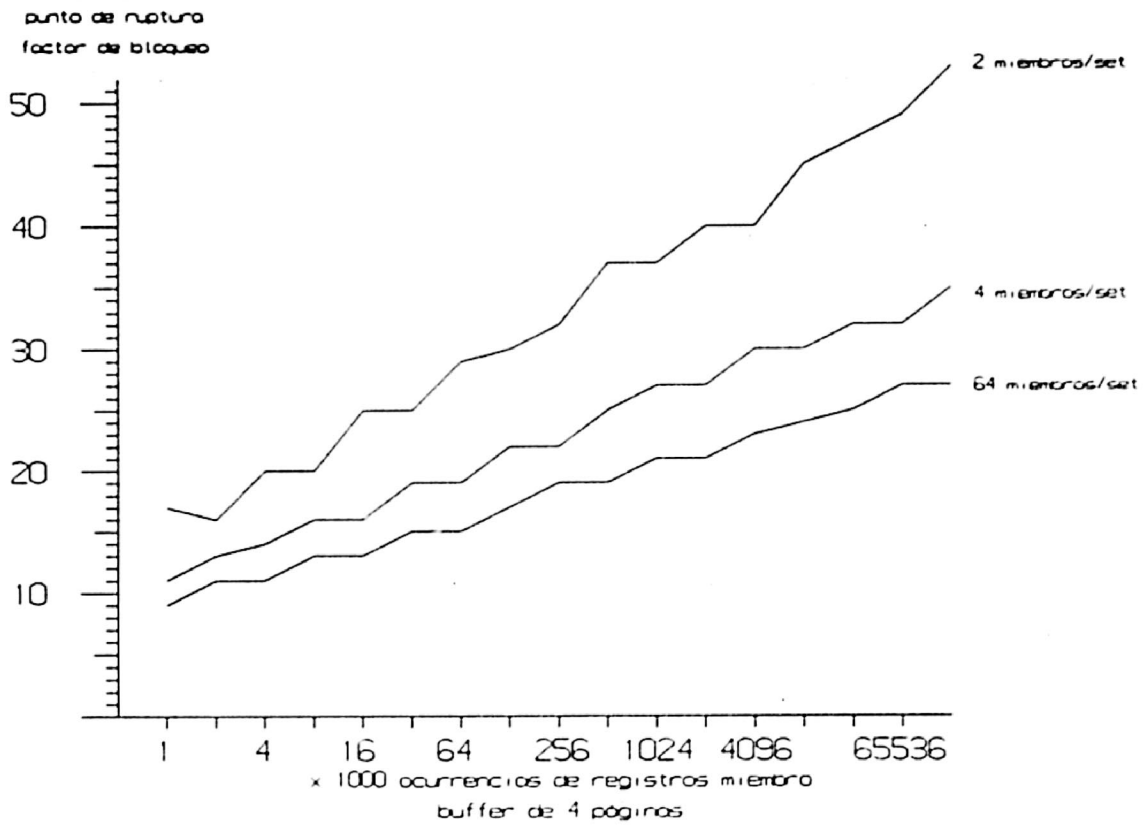
Figura 3.10

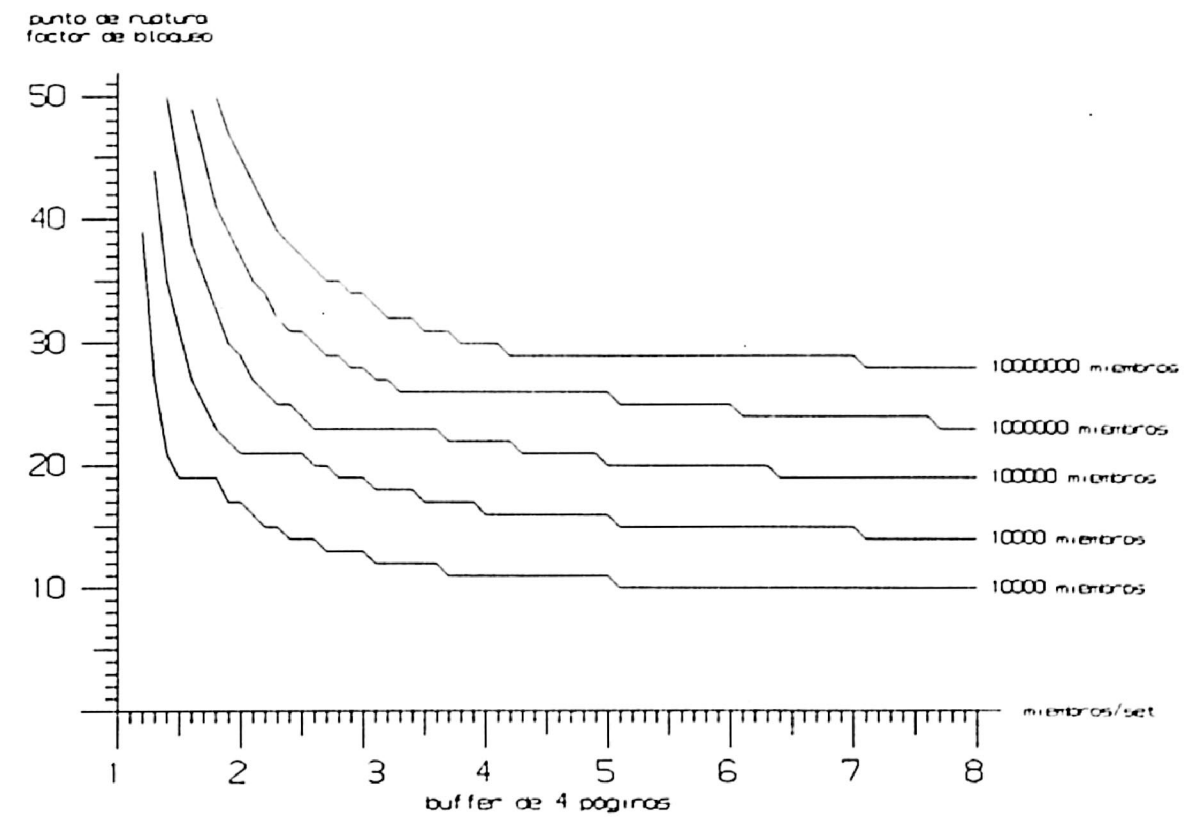
Comparación del método de clasificación de propietario con el de recuperación por tuplas.

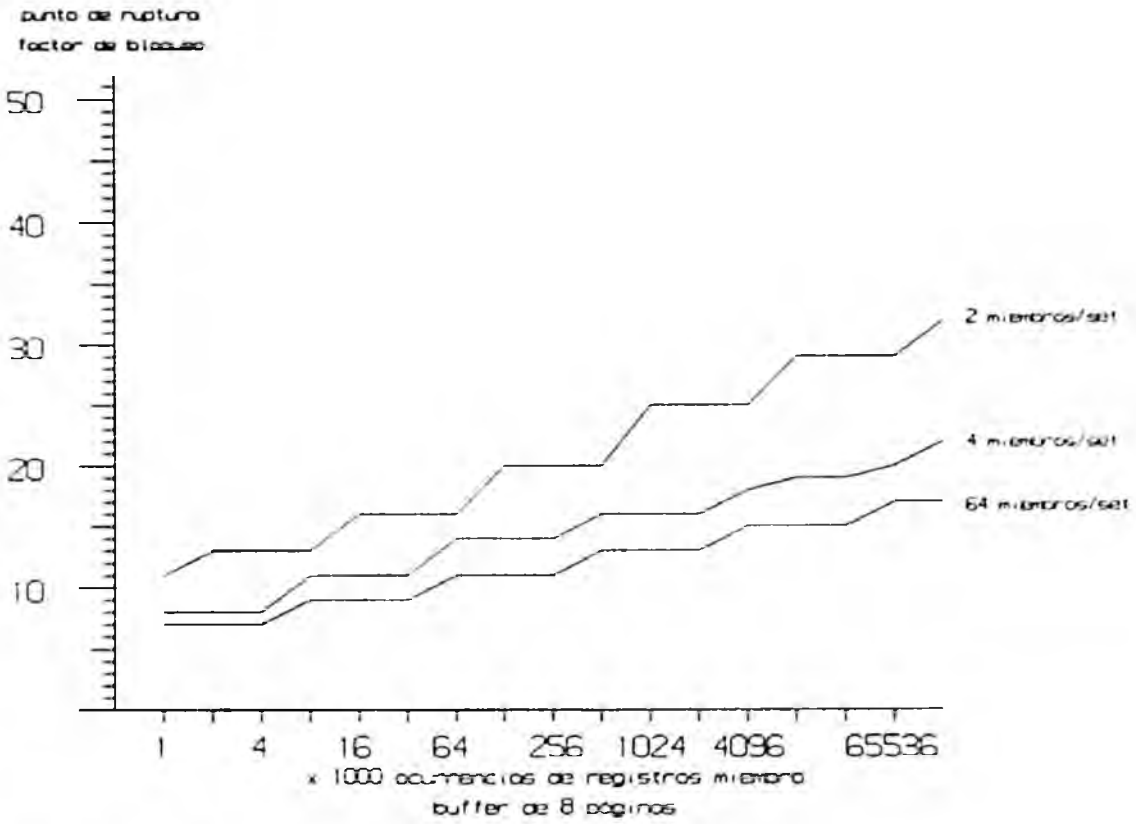
La figura 3.10 muestra que cuando las ocurrencias de set tienen dos ocurrencias miembro que satisfacen las condiciones de selección, y una página de disco puede contener por lo menos 28 tuplas de la relación de join, el método de clasificación de propietario con una relación en serie tiene mejor rendimiento.

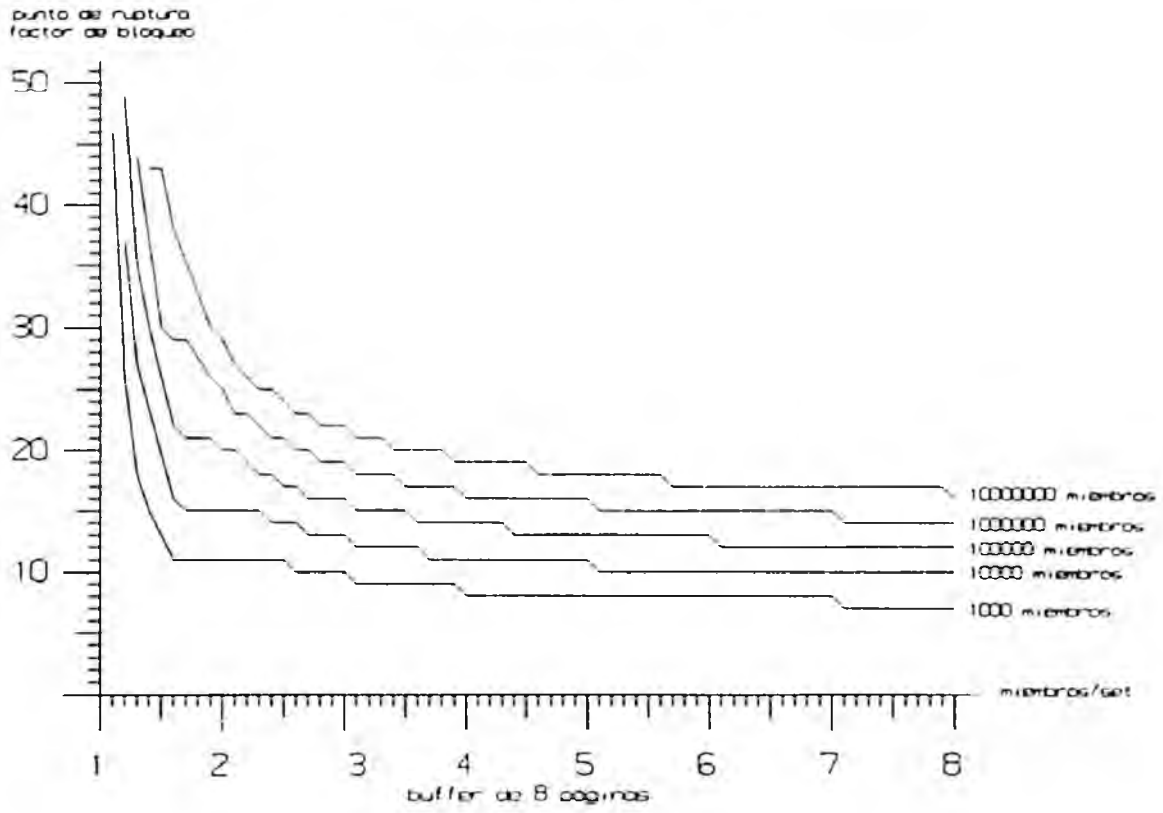
Llamaremos **punto de ruptura** al punto en el que el incremento de la velocidad es 1. La figura también muestra que los puntos de ruptura para 4 miembros/set y 16 miembros/set, son de 20 y 16 tuplas/página respectivamente.

La figura 3.11 representa los puntos de ruptura de diferentes casos. Por ejemplo, si tenemos un buffer de 8 páginas de memoria y cada ocurrencia de set tiene 2 ocurrencias miembro que satisfacen las condiciones, el punto de ruptura no es mayor de 32 tuplas/página, incluso cuando tenemos más de 65 millones de ocurrencias miembro.


punto de ruptura
factor de bloqueo
50
40
30
20
10
2 miembros/set
4 miembros/set
64 miembros/set
1
4
16
64
256
1024
4096
65536
x 1000 ocurrencias de registros miembro
buffer de 4 páginas


punto de ruptura
factor de bloqueo
50
40
30
20
10
10000000 miembros
1000000 miembros
100000 miembros
10000 miembros
1000 miembros
miembros/set
1
2
3
4
5
6
7
8
buffer de 4 páginas


punto de ruptura
factor de bloqueo
50
40
30
20
10
2 miembros/set
4 miembros/set
64 miembros/set
1
4
16
64
256
1024
4096
65536
x 1000 ocurrencias de registros miembro
buffer de 8 páginas


punto de ruptura
factor de bloqueo
50
40
30
20
10
10000000 miembros
1000000 miembros
100000 miembros
10000 miembros
1000 miembros
miembros/set
1
2
3
4
5
6
7
8
buffer de 8 páginas

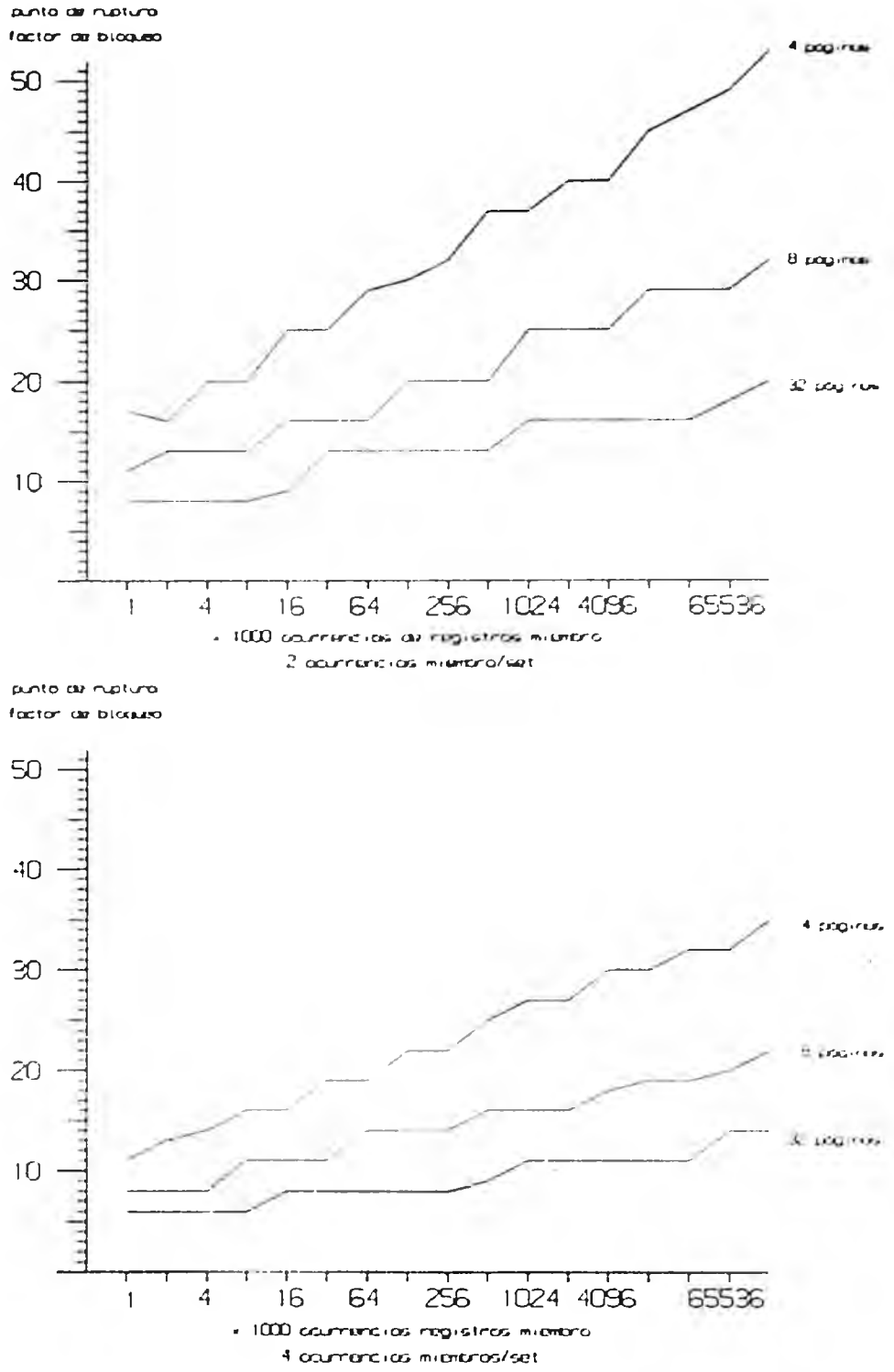


Figura 3.11

Puntos de ruptura para el método de clasificación de propietario con una relación en serie.

Siempre que hay una relación 1:1 entre propietario y miembro en el proceso de un query, con el método de recuperación por tuplas no hay recuperaciones redundantes de ocurrencias propietario, y por tanto la optimización no es necesaria. Cuando el factor de bloqueo es demasiado bajo, el tiempo para salvar, clasificar y recuperar la relación de join es mayor que el necesario para recuperar múltiples ocurrencias propietario. Ambos casos son excepciones.

Frecuentemente, se da el caso de que un query tiene una lista objeto pequeña y que para la selección requerida se compara el valor de algunos atributos existentes en el mismo tipo de registro. En consecuencia, el tamaño de cada tupla de las relaciones de 'join' y/o 'mant' es pequeño y puede esperarse que en una página de disco quepan más de 50 tuplas de relaciones intermedias.

La figura 3.12 muestra el número de accesos a páginas de disco necesarios para responder a tres queries Q1, Q2, y Q3, usando los métodos vistos (se asume que tenemos un buffer disponible de 5 páginas).

Sea Q1 el query siguiente:

> "Para cada suministrador, obtener todas las piezas que proporciona".

En este caso, el método de clasificación de propietario con una relación en serie es el mejor y muestra una mejora del 40% sobre el método de recuperación por tuplas.

Para Q2, el segundo query,

> "para cada representante de ventas, obtener, el número de pieza, el precio por unidad y la cantidad de todas las piezas que ha vendido a cada cliente",

el método de clasificación de propietario en paralelo es el mejor, en un 51%, que el método de recuperación por tuplas.

Para el último query,

> "para cada proveedor de Bilbao o de Vitoria, obtener el número de pieza y precio por unidad de todos los tornillos que proporciona",

el método que lo responde con el menor número de accesos a páginas de disco es el método de división de árbol con un rendimiento 44% mejor que el de recuperación por tuplas.

METODOS / QUERIES	Por tuplas	Clasificación propietario			División de árbol
		1 relac	2 relac	Paralelo	
Q1:DISPLAY S_NOM, P_NOM	20.020	12.082	12.600	12.600	14.820
Q2: DISPLAY R_NOM, C_NOM P#, PRECIO_U,CANT	600.100	315.116	295.628	293.751	310.402
Q3: DISPLAY S_NOM, P#, PRECIO_U WHERE (S_CIUDAD="Bilbao" OR S_CIUDAD="Vitoria") AND P_NOM="Tornillo"	5.020	3.800	3.656	3.656	2.811

Figura 3.12

Coste de evaluación de los queries

con los diferentes métodos.

Los métodos que damos para reducir el coste de recuperación de un registro como un propietario de un set, en general, son superiores al método tradicional, aunque no se puede afirmar que uno es siempre el mejor.

El método de recuperación por tuplas es el mejor cuando un set representa una relación 1:1 en vez de la relación general de 1:M. En el caso de relaciones 1:M, el método de clasificación de propietario es claramente superior, siempre que existan punteros 'owner' y no haya índices densos que hagan del método de división de árbol una alternativa factible.

El método de clasificación de propietario con una relación en serie requiere una clasificación por cada tipo de registro propietario que recuperamos. Si se utilizan dos relaciones en serie, se requieren dos clasificaciones y una combinación por cada registro tipo propietario que recuperamos. En paralelo, requiere dos clasificaciones por cada propietario, pero únicamente una interoalación por todo el query.

La diferencia entre las estrategias radica en los atributos que hay en los esquemas de relación intermedios que producen. Aplicando la estrategia en paralelo o la de dos relaciones en serie, puede a veces reducirse el número de atributos en cada resultado intermedio simple, por lo que las dos clasificaciones tardarán menos que la requerida en la estrategia de una relación en serie. Cuando no hay una lista objeto de atributos o atributos envueltos en selecciones para posterior evaluación, es dificil de superar el tiempo requerido por una estrategia de una relación en serie (un ambiente multiproceso podría hacer más deseable la estrategia en paralelo, por reducir el tiempo, incluso aunque el tiempo total fuese mayor que con una evaluación en serie). El método de división de árbol es la mejor estrategia siempre que no se disponga de punteros 'owner' o haya pocas ocurrencias tipo propietario que satisfagan las selecciones.

Los métodos dados pueden aplicarse recursivamente para cada subárbol que creamos y pueden combinarse libremente entre sí, y con otras técnicas de optimización de red y relacionales, para la evaluación de un query.

Ejemplo 3.7

Sea el esquema de red de la figura 2.4 y el query

DISPLAY C_NOM, S_NOM, S_CIUDAD.

El árbol de acceso contiene los tipos de registro C, O, S y R y los sets C-O, R-O, Y S-R. Para evaluar este query, primero recuperamos el tipo de registro O mediante una búsqueda secuencial en el área y dejamos el resultado en la relación $rjoin_1$ sobre $Rjoin_1$(R#,S#,C#). Entonces clasificamos $rjoin_1$ por R# S#, y recuperamos los tipos de registro R y S dejando el resultado en $rmant_2$ y $rjoin_2$ sobre $Rmant_2$(S_NOM,S_CIUDAD) y $Rjoin_2$(C#,INDICE), respectivamente. A continuación, clasificamos la relación $rjoin_2$ por C# y recuperamos los registros C, obteniendo rmant sobre Rmant(C_NOM,INDICE). Clasificamos rmant por INDICE y la combinamos con $rmant_2$ para responder al query. Esto indica que R y S se recuperan por el método de clasificación de

propietario, con una relación en serie, ya que el registro O no tiene atributos en la lista objeto, y C se recupera por el método de clasificación de propietario, con dos relaciones en serie, ya que S tiene dos atributos S_NOM y S_CIUDAD en la lista objeto.

CAPITULO 4

OPTIMIZACION DE QUERIES PARA BASES DE DATOS EN RED

4.1 INTRODUCCION

La optimización de queries para sistemas de bases de datos en red puede hacerse manualmente, por el usuario, o bien automaticamente, por un interfaz.

Un interfaz, con un lenguaje de interrogación de alto nivel para sistemas de bases de datos en red navegacionales, permite a los usuarios ignorar la organización física de la bases de datos y los caminos de acceso disponibles. Los usuarios formulan las preguntas describiendo meramente "qué" dato desean recuperar y no "cómo" recuperarlo. El interfaz traduce las preguntas de usuario en queries de red eficientes, seleccionando automáticamente, un camino de acceso óptimo para su proceso.

Para optimizar un query, hay que tener en cuenta que el número total de accesos a disco para procesarlo es clave en el rendimiento del interfaz.

Kuck y Sagiv, e Illarramendi utilizan un algoritmo heurístico para optimizar la tradución de queries de usuario [KUCK82, ILLA87]. Dayal y Goodman [DAYA82] muestran el mejor camino para responder a un query de árbol usando una base de datos en red. Pero ambas investigaciones asumen que siempre deben seguirse los punteros.

Por otra parte, un query no procedural, generalmente puede responderse correctamente evaluando uno de los diferentes árboles de acceso posibles; sin embargo, Dayal y Goodman sólo consideran un árbol de acceso para encontrar un camino de acceso óptimo.

A continuación, presentamos los algoritmos de optimización, que localizan un camino de acceso óptimo para un árbol de acceso dado, y un algoritmo, que enumera todos los árboles de acceso posibles para responder un query.

4.2 OPTIMIZACION DEL CAMINO DE ACCESO PARA UN ARBOL DE ACCESO

Sea un árbol de acceso T. Un camino de acceso para T prescribe el orden en que se pasa por los tipos de registro de ese árbol y puede expresarse mediante una secuencia de accesos a registro $(R_{i1},M_{i1}),\ldots,(R_{in},M_{in})$. Cada par (R_{ij},M_{ij}) representa un acceso a registro donde R_{ij} es el tipo de registro al que se accede, y M_{ij} es el método utilizado para el acceso. El primer registro de esta secuencia se llama registro cabecera del camino de acceso. Cada registro del árbol de acceso aparece exactamente una vez en el camino de acceso, y con la excepción del registro de partida no puede pasarse por ningún registro a menos que se haya hecho por un registro adyacente.

Llamaremos camino prefijo a una secuencia contigua de accesos a registro encabezada por (R_{i1},M_{i1}). Por ejemplo, $(R_{i1},M_{i1}),\ldots,(R_{ik},M_{ik})$, para $k\leq n$, es un camino prefijo del camino de acceso $(R_{i1},M_{i1}),\ldots,(R_{in},M_{in})$; sin embargo, la secuencia $(R_{i2},M_{i2}),\ldots,(R_{ik},M_{ik})$ no lo es.

Dado un camino prefijo P para un árbol de acceso T, un tipo de registro R_i es adyacente a P si hay un puntero o enlace S en T, entre R_i y R_j, y P contiene R_j pero no R_i.

4.2.1 ARBOLES DE DECISION

La programación dinámica es un método de diseño de algoritmos que puede usarse cuando la solución a un problema es el resultado de una secuencia de decisiones. Las elecciones posibles dependen del estado actual del sistema; es decir, de las decisiones previas.

En el problema de optimización de caminos, la decisión en cada paso es qué tipo de registro debe añadirse al camino prefijo y cuál es la forma más rápida de recuperar ese tipo de registro, dado el camino prefijo.

Un árbol de decisión se construye como sigue:

Partiendo del nudo raíz, por cada tipo de registro del árbol de acceso, se crea un nudo y un enlace de la raíz a ese nudo. Cada nudo consiste en un camino prefijo con su coste y tamaño. El coste es el número esperado de accesos a disco, requerido para evaluar el query de red, para el camino prefijo; y el tamaño es el número esperado de tuplas del resultado inmediato de dicho camino que satisface las condiciones de selección del query. Lógicamente sólo podrán aplicarse las selecciones en las que todas sus variables están en el camino prefijo.

Extendemos el árbol de decisión añadiendo al camino prefijo de cada nudo, un tipo de registro adyacente a él y así sucesivamente, hasta que todos los tipos de registro del árbol de acceso estén en el camino de acceso.

El coste del camino prefijo en un nudo del árbol de decisión se calcula sumando el coste esperado para recuperar el nuevo registro al coste del camino prefijo del nudo padre. El coste del nudo raíz es cero, y obviamente, el costo de cada nudo del árbol de decisión se calcula una sola vez.

Un transversal sobre el árbol de decisión es exactamente un algoritmo de búsqueda para optimizar el camino de acceso.

Una forma de encontrar el camino de acceso óptimo es mediante una búsqueda exhaustiva; es decir, enumerando todos los caminos de acceso, calculando el coste de cada uno de ellos y eligiendo el más económico.

Sea el camino prefijo $P_k=(R_{i1},M_{i1}),\ldots,(R_{ik},M_{ik})$; si $P_{k+1}=P_k \| (R_{ik+1},M_{ik+1})$, entonces el coste esperado de P_{k+1} es mayor o igual al de P_k. Cuando el coste tiene esta propiedad, la técnica conocida como "branch and bound" [REIN77] puede aplicarse para evitar la generación de P_{k+1}, cuando el coste de P_k es mayor o igual que el coste óptimo actual.

4.2.2 METODO DE RECUPERACION POR TUPLAS

El apéndice C.1 muestra el algoritmo ALG1. Este algoritmo localiza el camino de acceso óptimo para un árbol de acceso mediante el método de recuperación por tuplas. ALG1 tiene como entrada el árbol de acceso T.

El coste óptimo es una variable positiva que puede tomar cualquier valor y se inicializa con el valor mayor posible. A continuación, se empiezan a construir caminos de acceso, creando un camino prefijo para cada tipo de registro R de T. El tipo de registro R de cabecera puede recuperarse usando uno de los métodos siguientes:

1.- Mediante una búsqueda secuencial en todas las áreas que pueden contener ocurrencias de R.
2.- Mediante una clave de búsqueda, si R es un registro miembro de un set singular, que tiene definida una clave de búsqueda y en el query se han dado valores para esa clave.
3.- Como un registro miembro de un set singular, que contenga todas las ocurrencias de tipo R sin usar un índice, si R es un registro miembro de un set singular.
4.- Por medio de una clave de cálculo, si está definida para R y en el query se dan los valores de los atributos que forman dicha clave.

5.- Si $R_1,\ldots,R_k,R$ es un camino dirigido del esquema de red N, tal que R_1 es un tipo de registro miembro de un set singular y ninguno de los tipos de registro $R_1,\ldots,R_k$ aparece en T, entonces para recuperar R puede aplicarse el camino prefijo:
$P=(R_1,\text{set singular})\|\ldots\|(R,\text{miembro del set } S_k)$.

Se calcula el coste esperado para acceder a R usando cada uno de los posibles métodos y se elige el de menor coste, M. Este coste de recuperación de R mediante M con el camino prefijo dado, se calcula con las fórmulas desarrolladas en la sección 3.5.

Si el coste esperado para acceder a R es mayor o igual que el coste óptimo actual, entonces este camino prefijo ya no se ampliará más.

En caso contrario, se crea un camino prefijo (R,M) (si el método 5 es el más económico para recuperar R, entonces se crea el camino prefijo $P=(R_1,\text{set singular})\|\ldots\|(R,\text{miembro del set } S_k)$ en lugar de (R,M)). Si T tiene un solo tipo de registro, se actualiza el coste óptimo y (R,M) será ahora el camino de acceso óptimo; en caso contrario, entramos en una fase iterativa.

En cada iteración extendemos el árbol prefijo P con un tipo de registro hasta que aparezcan en el camino de acceso todos los registros de T, o bien, el coste del camino prefijo sea mayor o igual que el coste óptimo en ese momento.

Cada registro R diferente del de cabecera del camino prefijo, se recupera como miembro de un set S o bien como propietario de S, donde S es el set que enlaza R con un registro que está antes de R en el camino de acceso.

Si R se recupera como miembro de S, entonces R puede recuperarse mediante una clave de búsqueda si los valores de los atributos que forman esa clave se dan en el query. En caso contrario, tendrán que recuperarse todas las ocurrencias miembro de la ocurrencia de set.

Si el coste total actual es menor que el óptimo, entonces, se añade al camino prefijo (R,M), siendo M el método más económico para recuperar R.

Si todos los registros de T aparecen en el camino de acceso P y el coste esperado de P es menor que el coste óptimo actual, entonces, se actualiza el coste óptimo y P es ahora el camino de acceso óptimo.

Ejemplo 4.1

Sea el esquema de red de la figura 2.4 y el query

```
DISPLAY S_NOM, P#, PRECIO_U
   WHERE (S_CIUDAD ="VITORIA" OR
          S_CIUDAD="BILBAO")
     AND P_NOM="TORNILLO"
```

Supongamos que 5 de los 20 suministradores están en Vitoria o Bilbao y 400 de las 1000 piezas son "tornillos".

El árbol de acceso utilizado para responder este query contiene los tipos de registro S, P, y SP y los sets S-SP y P-SP.

Para encontrar el camino de acceso óptimo usando el algoritmo ALG1, empezamos creando un camino prefijo para cada uno de los tipos de registro S, P y SP, y así, localizar todos los caminos de acceso posibles. S puede ser recuperado como registro miembro de un set singular con Cost=20 accesos a disco y Tamaño=5 tuplas. Ya que todos los tipos de registro del árbol de acceso no están en el camino prefijo y SP es el único registro adyacente a él, lo extendemos añadiéndole SP. Este, se recuperará como registro miembro del set S-SP con Cost=2520 accesos y Tamaño=2500 tuplas. Añadimos P al camino prefijo y lo recuperamos como propietario del set P-SP. Para el

camino de acceso S,SP,P, el coste es de 5.020 accesos, que es menor que el coste óptimo actual (el coste óptimo actual se inicializa como infinito). Por tanto el camino de acceso óptimo actual ahora es S,SP,P y el coste óptimo actual 5.020 accesos a disco. Las mismas operaciones se llevan a cabo con los árboles de acceso encabezados por los registros P y SP.

La figura 4.1 muestra la parte del árbol de decisión que es utilizado durante la ejecución de ALG1. Cuando ALG1 termina, obtenemos el camino de acceso óptimo S,SP,P y el coste óptimo 5020 accesos a páginas de disco.

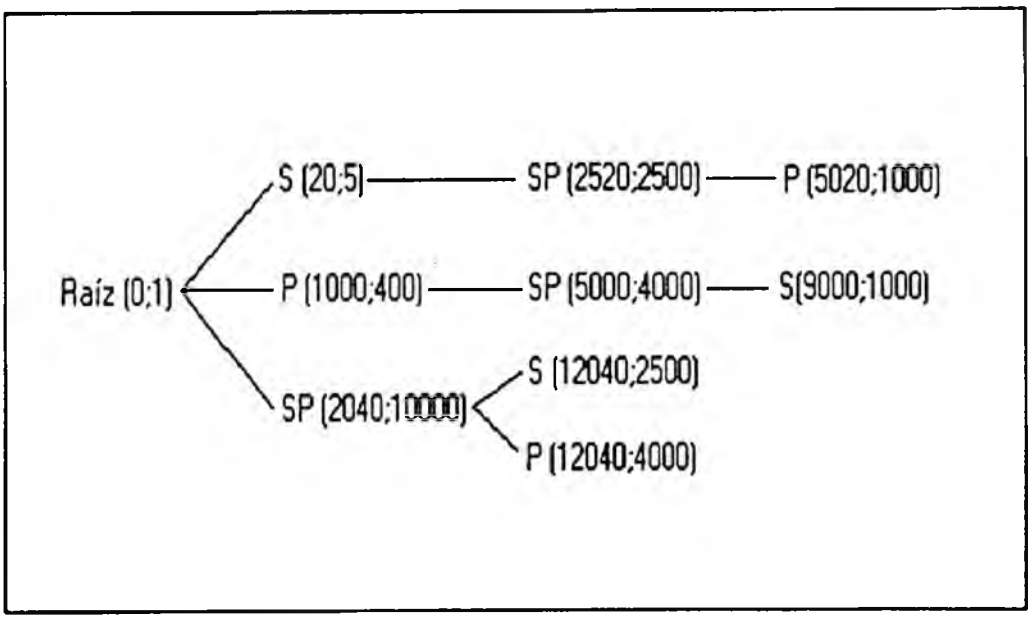


Figura 4.1

Arbol de decisión para la ejecución de ALG1.

4.2.3 COMBINACION DE LOS METODOS DE RECUPERACION POR TUPLAS Y DE CLASIFICACION DE PROPIETARIO

Sea un árbol de acceso T y To, un subárbol de T que contiene todos los tipos de registros y sets que están en el camino prefijo Po.

Como se muestra en la figura 4.2, To está conectado con los registros $Ro_1, Ro_2, \ldots, Ro_n$, por los sets $So_1, So_2, \ldots, So_n$, pertenecientes a T. To también está conectado en T, por los sets $Sm_1, Sm_2, \ldots, Sm_p$, a $Rm_1, Rm_2, \ldots, Rm_p$, respectivamente. Cada Ro_k ($1 \leq k \leq n$) es el tipo de registro propietario del set So_k y cada Rm_j ($1 \leq j \leq p$) es el tipo de registro miembro del set Sm_j.

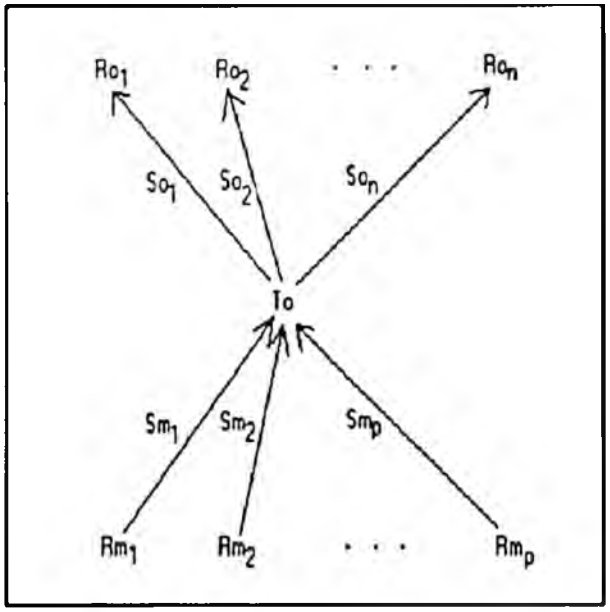


Figura 4.2

Tipos de registro conectados a To en T.

Supongamos que algún Ro_k ($1 \leq k \leq n$) es el siguiente tipo de registro a añadir al camino prefijo Po. Ro_k se recuperará como propietario del set So_k.

Combinando los métodos de recuperación por tuplas y de clasificación de propietario, Ro_k puede recuperarse siguiendo una u otra estrategia. El apéndice C.2 muestra la parte iterativa del algoritmo ALG2 que localiza un camino de acceso óptimo para implementar un árbol de acceso, combinando los métodos de recuperación por tuplas y de clasificación de propietario.

Lo que distingue ALG2 de ALG1 es que, cuando se recupera un registro como propietario de un set, también se consideran como alternativas los dos métodos de clasificación de propietario en serie. Es decir, en el paso iterativo de ALG2, si el tipo de registro R se puede recuperar como propietario del set S, calculamos los costes para los dos métodos de clasificación de propietario en serie, así como para el de recuperación por tuplas. Se elige el método que produce menor número de accesos a disco.

Si el tipo de registro Ro_k, de la figura 4.2, va a recuperarse como propietario del set So_k por el método de clasificación de propietario con una relación en serie, el esquema de la relación intermedia (relación de join)

contendrá los atributos de la lista objeto, los atributos que aparecen en las selecciones junto con atributos no pertenecientes a To, una clave por cada tipo de registro Ro_j ($1 \leq j \leq n$) y la clave del registro propietario de cada set Sm_j ($1 \leq j \leq p$).

El coste de recuperar Ro_k con este método viene dado por el de salvar, clasificar y recuperar la relación intermedia más el coste de recuperar una sola vez cada ocurrencia propietaria Ro_k de interés.

Cuando se usa el método de clasificación de propietario con dos relaciones en serie para recuperar Ro_k, se generan dos relaciones intermedias: rmant y rjoin.

El esquema de relación Rmant para rmant contiene los atributos de la lista objeto que están en To, los atributos presentes en las selecciones junto con atributos que no están en Ro_k o en To, una clave de cada registro de tipo Ro_j ($1 \leq j \leq n, j \neq k$) y la clave del registro propietario de cada set Sm_j ($1 \leq j \leq p$).

El esquema de relación Rjoin para rjoin contiene una clave de Ro_k, los atributos de las selecciones con atributos de Ro_k y un atributo INDICE.

El esquema de relación Rcombi para rcombi, resultado de recuperar Ro_k, contiene los atributos de la lista objeto, los atributos de las selecciones con atributos que no están en To o en Ro_k y el atributo INDICE. Rcombi también contiene una clave de cada tipo de registro propietario que está conectado a Ro_k por un set que no está en To o So_k.

El coste de recuperar Ro_k con este método, es la suma de los siguientes costes: almacenar y recuperar rmant; almacenar, clasificar y recuperar rjoin; almacenar, clasificar y recuperar rcombi; y recuperar cada ocurrencia propietaria Ro_k sólo una vez.

La parte restante de ALG2 es la misma que la de ALG1. Este pequeño cambio podría darnos una gran reducción de coste, para incluso un query simple.

Ejemplo 4.2

Sea el esquema de red de la figura 2.4 y el query del ejemplo 4.1. Para encontrar el camino de acceso óptimo con el algoritmo ALG2, empezamos construyendo caminos de acceso creando un camino prefijo por cada uno de los tipos de registro S, P, y SP.

El registro S puede recuperarse como miembro de un set singular con Cost=20 accesos a disco y Tamaño=5 tuplas. Como en el camino prefijo no están todos los registros del árbol de acceso y SP es el único registro que es adyacente a él, extendemos el camino prefijo añadiendo SP. Este registro se recuperará como registro miembro del set S-SP con Cost=2520 accesos y Tamaño=2500 tuplas.

Finalmente, añadimos P al camino prefijo y lo recuperamos como propietario del set P-SP.

Supongamos que en memoria tenemos un buffer de 5 páginas. El coste para el camino de acceso S,SP,P usando el método de recuperación por tuplas, el declasificación de propietario con una relación en serie y el de clasificación de propietario con dos relaciones en serie será 5020, 3800 y 3656 accesos a disco, respectivamente.

A la vista de los resultados, elegimos el método de clasificación de propietario con dos relaciones en serie para recuperar P, y actualizamos el camino de acceso óptimo y el coste óptimo.

La figura 4.3 muestra la parte del árbol de decisión que se utiliza en la ejecución de ALG2, para el query del ejemplo. Cuando ALG2 finaliza, obtenemos

el camino óptimo: S,SP,P y el coste óptimo: 3656 accesos a disco que presenta una mejora del 27% sobre el coste óptimo del algoritmo ALG1.

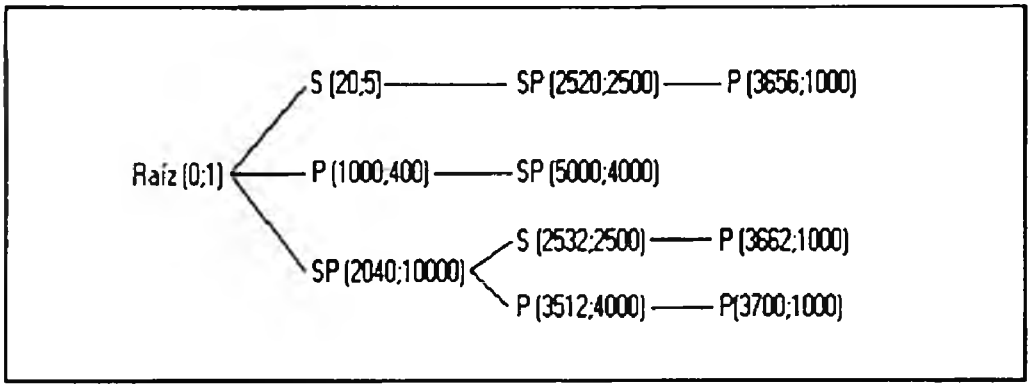


Figura 4.3

Arbol de decisión para ejecutar el algoritmo ALG2.

Si la base de datos se almacena en dos áreas o más en lugar de una, tal que los registros S, P y SP se almacenen en una y el resto de registros en otras, entonces la búsqueda secuencial en todas las áreas en las que puede haber ocurrencias de SP puede ser mucho más eficiente. El camino de acceso óptimo será SP,S,P, y el coste óptimo 1696 accesos a disco (un 66% mejor que el coste de ALG1).

Supongamos que el árbol de acceso T contiene n tipos de registros. En los algoritmos ALG1 y ALG2, el camino prefijo se va extendiendo con un registro en cada

iteración. Cada rama del árbol de decisión contendrá como máximo n nudos; es decir, como máximo habrá n pasos de decisión. El primer paso tiene n posibles elecciones; es decir, el primer nivel del árbol de decisión tiene n ramas. En el paso i ($2 \le i \le n$), el nivel i-1 del árbol de decisión tiene como máximo n x (n-1) x···x (n-i+2) ramas. Cada rama del paso i-1 no tiene más de (n-i+1) opciones posibles. Por tanto, como máximo hay n! ramas en el paso n; es decir, las n! permutaciones de los n tipos de registro. El árbol tiene como máximo:

$$\sum_{i=1}^{n} \binom{n}{i} = O(n \times n!) \text{ nudos.}$$

Por tanto, el tiempo de cálculo de la complejidad de los algoritmos ALG1 y ALG2 es O(n x n!). En realidad, no pueden usarse como camino de acceso todas las ordenaciones de n tipos de registro. En cada paso se aplica la técnica "branch and bound" para preservar la expansión del camino prefijo, si no es más económico que el coste óptimo actual. Incluso aunque la complejidad del tiempo de cálculo sea exponencial, dado que el número de registros envueltos en el query es en general suficientemente pequeño, el tiempo para ejecutar cualquiera de estos algoritmos es aún razonable [SELI79, DAYA82].

En los algoritmos ALG1 y ALG2 como máximo aparecen en un momento dado n nudos del árbol de decisión. Cada nudo contiene un camino prefijo P (es decir, una secuencia de accesos a registros), el coste Cost, y el tamaño Tamaño,

consumiendo O(n) espacio de memoria. Por tanto, la fórmula del espacio de memoria necesario para los algoritmos ALG1 y ALG2 es $O(n^2)$.

4.2.4 OPTIMIZACION GLOBAL

Sea el árbol de acceso T, R el tipo de registro de T que no tiene descendientes, y $Ro_1,\ldots,Ro_m$ los registros conectados a R por los sets $So_1,\ldots,So_m$ como se muestra en la figura 4.4.

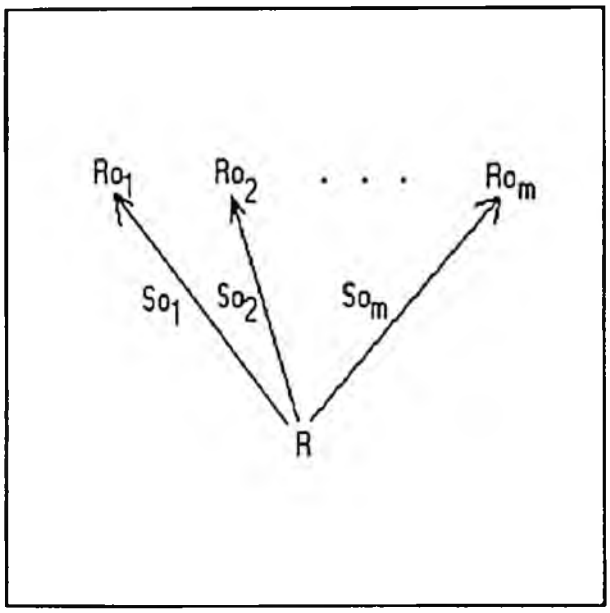


Figura 4.4

Registros conectados a R en T.

Si eliminamos los enlaces So_j de T entre R y Ro_j para todo j ($1 \leq j \leq m$), tendremos m+1 subárboles. Es decir, un subárbol T_1, ..., T_m para cada registro Ro_1, ..., Ro_m propietario y otro subárbol para R.

La figura 4.5 muestra cómo se conectan entre sí en el árbol de acceso T, los subárboles que creamos.

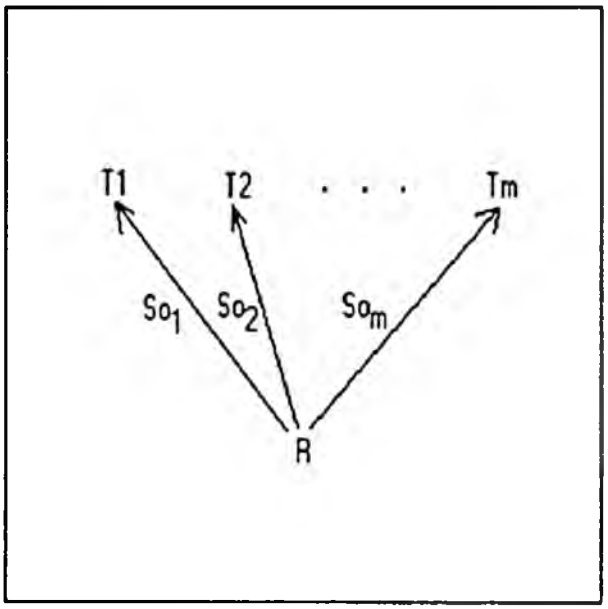


Figura 4.5

Conexiones entre los subárboles creados en T.

Para evaluar un query mediante T, se pueden usar diferentes aproximaciones:

- Podemos recuperar primero R y luego evaluar cada subárbol T_i ($1 \leq i \leq m$) hacia arriba (es decir, recuperando todos sus ancestros como propietarios de un set).

- O bien, podemos evaluar primero uno o más subárboles por el método de división de árbol, combinar las relaciones intermedias, y después evaluar los subárboles restantes hacia arriba.

El apéndice C.3 muestra el algoritmo ALG3. Este algoritmo localiza un camino de acceso óptimo para implementar un árbol de acceso T, combinando los métodos de recuperación por tuplas, el de clasificación de propietario y el de división de árbol.

Primero aplicamos el algoritmo ALG2 para localizar un camino de acceso óptimo, combinando el método de clasificación de propietario en serie y el tradicional.

Si T contiene sólo un tipo de registro, el algoritmo finaliza. Si no, aplicamos el método de división de árbol al registro R que no tiene descendientes para descubrir si hay algún camino de acceso menos caro que el óptimo localizado con el algoritmo ALG2.

Iniciamos la búsqueda de un camino de acceso óptimo P_i, para cada subárbol T_i ($1 \le i \le m$) que está conectado con R por el set So_i, aplicando recursivamente el algoritmo ALG3. Para cada subárbol T_i ($1 \le i \le m$) se añade al camino prefijo P_i, el método de recuperación de R, o bien del índice denso para las ocurrencias miembro de una ocurrencia del set So_i. El esquema de relación $Rmant_i$, resultante de la evaluación de P_i ($1 \le i \le m$), contiene la lista objeto de atributos, los atributos de las selecciones con atributos que no están en P_i y una clave de R. La clave de R se obtiene recuperando R o un índice denso para las ocurrencias miembro de la ocurrencia del set So_i. Para evaluar cualquier combinación de k ($1 \le k \le m$) subárboles podemos aplicar el método de división de árbol y aplicar otros métodos para evaluar los subárboles restantes.

En otras palabras, podemos evaluar cualquier conjunto de k caminos prefijos en paralelo o en cualquier orden; clasificar las k relaciones de mantenimiento por la clave de R; combinar estas relaciones mediante la clave de R; recuperar R usando su clave, cuando sea necesario; y entonces, recuperar los subárboles restantes por el método tradicional de recuperación por tuplas o por los de clasificación de propietario en serie.

Dado que R es miembro de m sets diferentes, tenemos 2m-1 combinaciones posibles a comprobar. La parte de 'proceso hacia arriba' bautizada como UP de ALG3 es

similar a la parte iterativa de ALG2 excepto en que cada registro sólo puede recuperarse como propietario de algún set del que sea propietario. Una vez terminado este proceso para una combinación específica de k subárboles, mediante el método de división de árbol, se recuperan los restantes por otros métodos, encontrándose así un camino de acceso óptimo para la combinación.

Si el coste óptimo de la combinación es menor que el que teníamos, entonces se actualizan tanto el coste óptimo como el camino de acceso óptimo actual. Cuando todas las posibles combinaciones para R se han comprobado, el algoritmo ALG3 termina obteniéndose el camino de acceso óptimo.

Ejemplo 4.3

Sea el esquema de red de la figura 2.4 y el query del ejemplo 4.1.

Para encontrar un camino de acceso óptimo mediante el algoritmo ALG3, aplicamos primeramente ALG2 que obtiene como camino de acceso óptimo S,SP,P con un coste de 3656 accesos a disco, tal como se describe en el ejemplo 4.2.

El tipo de registro SP no tiene registros descendientes y está conectado a los registros S y P por los sets S-SP y P-SP, respectivamente. Por tanto, aplicamos ALG3 a los subárboles que contienen S y P, respectivamente.

Recuperar S, requiere 20 accesos a disco y el resultado que se obtiene es de 5 tuplas. La recuperación de P requiere 1000 accesos a disco obteniéndose 400 tuplas.

Suponiendo que cada página puede contener unos 170 nudos de índices densos para los miembros de cada ocurrencia de los sets P-SP y S-SP, recuperar un índice denso para los sets S-SP y P-SP requiere 15 y 400 accesos a disco, respectivamente.

La relación $rmant_1$ sobre $Rmant_1$(S#,P#,S_NOM) para el subárbol de S tiene 2500 tuplas. Cada tupla de $rmant_1$ requiere 28 bytes de memoria. Suponiendo que tenemos un buffer disponible de 5 páginas de memoria, salvar, clasificar y recuperar rmant, requerirá 280 accesos a disco.

La relación $rmant_2$ sobre $Rmant_2$(S#,P#) para el subárbol de P tiene 400 tuplas. Cada tupla de $rmant_2$ requiere 8 bytes de memoria. Por tanto, para salvar, clasificar y recuperar $rmant_2$ se requerirán 96 accesos

a disco. El resultado de combinar $rmant_1$ y $rmant_2$ por S# P#, tiene 1000 tuplas por lo que recuperamos sólo 1000 ocurrencias de SP con un coste de 1000 accesos a disco.

El coste total del camino de acceso S,P,SP por el método de división de árbol es, por tanto, de 2811 accesos.

4.3 OPTIMIZACION DE UN QUERY SENCILLO

En esta sección describimos cómo optimizar un query Q, representado por un panel C con una sola fila.

El proceso de este tipo de queries tiene tres fases:

- Localización de todos los R-transversales que deben calcularse.
- Elección de un camino de acceso óptimo para cada R-transversal.
- Generación del código necesario para evaluar cada camino de acceso y su ejecución.

Lógicamente, el camino de acceso contiene todos los atributos A que la columna A del panel tiene como una variable diferenciada, como una constante, o como una variable no diferenciada que aparece en el bloque de condiciones.

El usuario formula preguntas o queries, con respecto a los atributos X de las relaciones que le presenta el interfaz en lugar de contra la bases de datos en red subyaciente.

Si $X \subseteq \bar{R}_i$, y no hay un registro ancestro R_j de R_i tal que $X \subseteq \bar{R}_j$, entonces por cada tipo de registro R_i del esquema de red, debe calcularse el R_i-transversal para responder al query.

Sea GR(VR,ER) un grafo tal que VR contiene el tipo de registro R y todos sus ancestros, y ER contiene todos los sets S_j del esquema de red que tienen un registro miembro en VR. Un árbol de acceso TR, para el R-transversal, es un subgrafo de GR, tal que TR es un árbol y todo atributo A de X aparece en algún tipo de registro de TR.

A continuación, mostramos cómo enumerar los árboles de acceso para un R-transversal dado y describimos la optimización del camino de acceso para él. Una vez hecho esto, mostramos cómo optimizar un query dado por un panel con una sola fila.

4.3.1 ENUMERACION DE LOS ARBOLES DE ACCESO PARA UN R-TRANSVERSAL

El apéndice C.4 muestra el algoritmo GRAFO que calcula el conjunto de todos los posibles árboles de acceso PT para un R-transversal dado, con respecto a X.

Dado un R-transversal de un esquema de red N y un conjunto de atributos X, que aparece en el panel del query, construimos primeramente un grafo GR(VR,ER), llamado R-grafo. VR contendrá R y todos sus ancestros; y para todos los sets S_j tal que R o un ancestro de R es miembro de él, S_j es un enlace en ER.

Inicializamos GR poniendo VR igual a {R} y ER igual al conjunto vacío.

Añadimos a ER todos los sets S_j de N que tienen a R como registro miembro y a VR todo ancestro de R en N, tipo R_i, y por cada R_i añadimos a ER todos los sets en los que es un registro miembro. Una vez construido el grafo R-grafo enumeramos todos los árboles de acceso.

Todo registro del R-grafo está también en el árbol de acceso T, pero el nudo hoja R_h de T cuyos atributos no están en X, se eliminará de dicho árbol, ya que no es necesario recuperar R_h para calcular el R-transversal para

X. Por cada registro tipo R_1 diferente del registro raíz R de GR que es propietario de k sets, se elige uno de esos k sets para conectarlo en T con su miembro.

Supongamos que GR contiene m tipos de registro $R, R_{i1}, R_{i2}, \ldots, R_{im-1}$. y cada R_{ij} $(1 \leq j \leq m-1)$ posee k_j sets; como máximo habrá $k_1 \times k_2 \times \cdots \times k_{m-1}$ árboles de acceso posibles. El algoritmo GRAFO los enumerará.

Sea un R-grafo que contiene n tipos de registro. Dado que GR no tiene ciclos, como máximo habrá (n-1)! árboles de acceso posibles y un enlace entre cualquier par de tipos de registros en GR.

En el caso de que cada registro de GR esté conectado a cada uno de sus ancestros por un enlace, entonces la raíz no será propietaria de ningún set, las hojas poseerán n-1 sets, y los registros restantes, n-2,...,2,1 sets respectivamente. Por tanto, como máximo también habrá (n-1)! árboles de acceso posibles para GR.

Ejemplo 4.4

Sea el esquema de red de la figura 4.6 y el query siguiente:

```
DISPLAY P WHERE K NOT= C
```

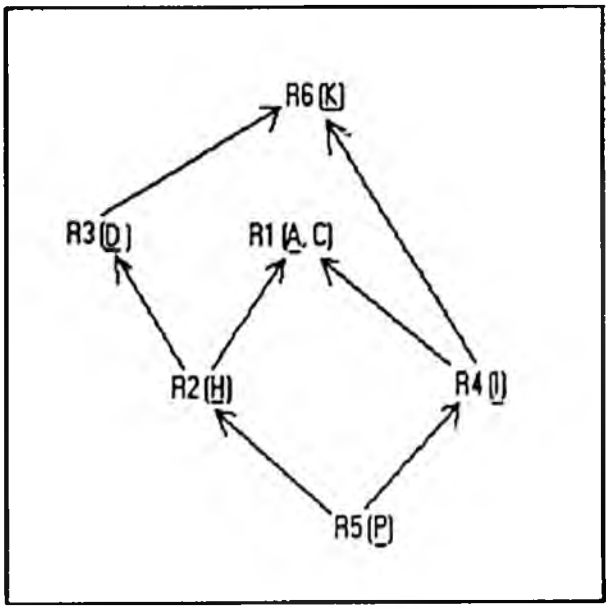


Figura 4.6

Esquema de red.

Para responderlo debemos calcular el R5-transversal. El esquema de figura 4.6 coincide con el R5-grafo.

Los tipos de registro R1 y R6 son propietarios de 2 sets, mientras que el resto de los registros sólo posee uno. Por tanto, tenemos cuatro árboles de acceso posibles a considerar para encontrar un camino de acceso óptimo para responder el query. Estos árboles de acceso se muestran en la figura 4.7.

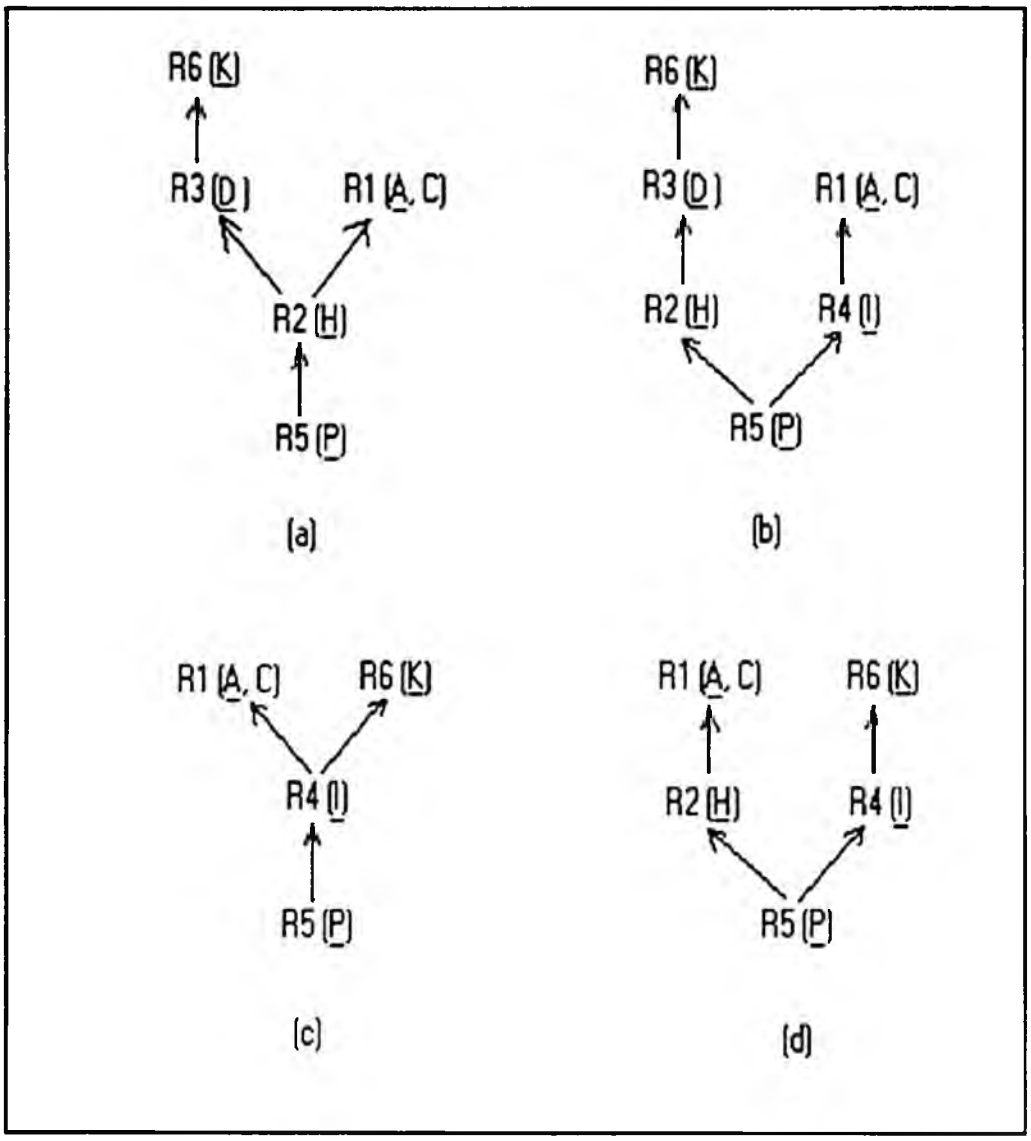


Figura 4.7

Arboles de acceso para el query.

Obsérvese que sólo el árbol de la figura 4.7(b) contiene los seis registros que aparecen en el R5-grafo. En la figura 4.7(c) por ejemplo, no aparecen los registros R2 y R3 en el árbol de acceso. En el query no aparece ningún atributo de R2 o de R3

y ninguno de ellos es parte de un camino en el árbol de acceso a sus ancestros, con algún atributo en el query. Por tanto, no es necesario recuperar ni R2 ni R3 para evaluar el query, usando el árbol de acceso.

4.3.2 OPTIMIZACION DEL CAMINO DE ACCESO PARA UN R-TRANSVERSAL

El apéndice C.5 presenta el algoritmo OPT-A que optimiza el camino de acceso para un R-transversal con respecto a X.

Dado un R-transversal para responder a un query referido a un conjunto de atributos X, tras inicializar el coste óptimo a $+\infty$, el algoritmo OPT-A construye el R-grafo y enumera el conjunto de todos los posibles árboles de acceso aplicando el algoritmo GRAFO.

Por cada árbol de acceso enumerado, se aplica el algoritmo ALG2 o el ALG3 para encontrar el camino de acceso óptimo. Si el coste de ese camino es menor que el coste óptimo actual para el R-transversal, se actualizan el camino de acceso y el coste óptimo para dicho transversal.

Una vez encontrado el camino de acceso óptimo para cada árbol de acceso, OPT-A termina habiendose se localizado el camino de acceso óptimo para el R-transversal.

Ejemplo 4.5

Sea el esquema de red de la figura 2.4 y el query siguiente:

DISPLAY S_NOM, C_NOM, P#, CANT

Esta petición puede responderse calculando el transversal de O. El O-grafo contiene los seis tipos de registro del esquema de red.

La figura 4.8 muestra los dos árboles de acceso posibles para este transversal.

El camino de acceso óptimo para el árbol de la figura 4.8(a) es:
((O, todas las áreas), (SP, clasificación de propietario set SP-O con una relación en serie), (S, propietario del set S-SP), (C, clasificación de propietario set C-O con una relación en serie)),
y el coste es de 91.384 accesos a disco.

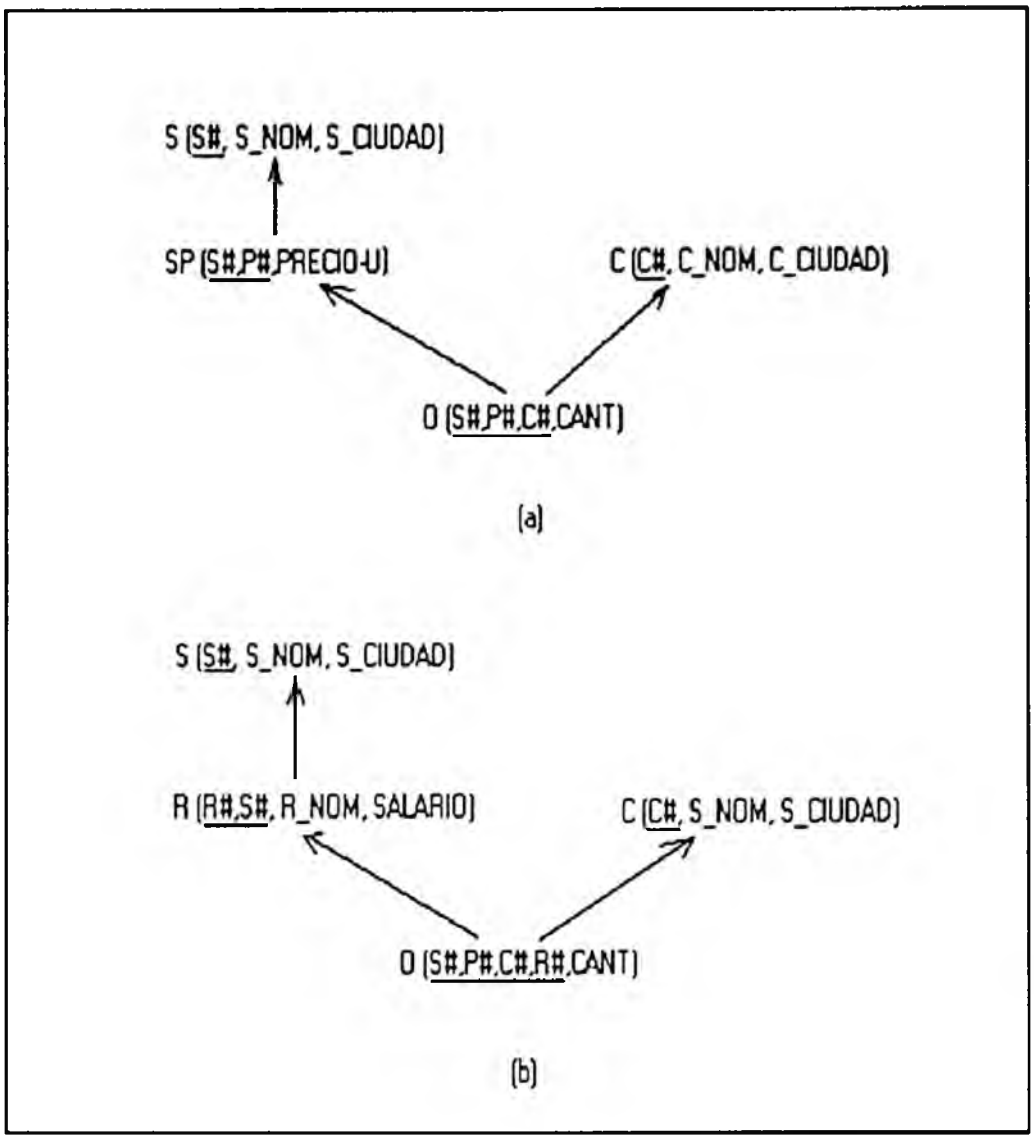


Figura 4.8

Arboles de acceso para responder al query

El camino de acceso óptimo para el árbol de acceso de la figura 4.8(b) es:

((O, todas las áreas), (R, clasificación de propietario set R-O con una relación en serie),

(S, propietario del set S-R), (C, clasificación de propietario set C-O con una relación en serie)),y el coste es de 71.584 accesos a disco. Por tanto, este último es el camino de acceso óptimo para responder al query.

Si el sistema de base de datos es globalmente consistente [KUCK83], el algoritmo ALG2 usado en OPT-A, puede integrarse formando un nuevo algoritmo que llamaremos OPT-B y que mostramos en el apéndice C.6.

En primer lugar, OPT-B calcula el R-grafo como es usual, pero enumerando todos los caminos de acceso posibles directamente del R-grafo. Esto permite aplicar antes el método 'branch and bound' consiguiéndose reducir en algunos casos el coste de optimización.

Sea GR(VR,ER) el R-grafo a partir del cual se calcula el R-transversal. Inicialmente, el coste óptimo se pone a infinito positivo y se empieza construyendo los caminos de acceso creando un camino prefijo para cada tipo de registro R_i de VR.

El registro de cabecera R_i puede recuperarse por uno de los métodos siguientes:

1.- Búsqueda secuencial en todas las áreas que pueden contener alguna ocurrencia de R_i.

2.- Recuperación como miembro de un set singular usando o no una clave de búsqueda.

3.- Por medio de una clave de cálculo.

Usando cada uno de los métodos posibles se calcula el coste esperado de acceso a R_i y se elige el más económico: M.

Si el coste esperado para recuperar R_i es mayor o igual que el coste óptimo actual, no se extiende más este camino. En caso contrario, se crea un camino prefijo y si todo atributo de X aparece en R_i, se actualiza el coste óptimo y (R_i,M) es el camino de acceso óptimo actual; si no entramos en una fase iterativa.

Para el paso iterativo, extendemos el camino prefijo Path con un acceso a registro en cada iteración hasta que todo atributo de X aparezca en el camino de acceso o el coste del camino prefijo sea mayor o igual que el coste óptimo actual.

Un tipo de registro R_k de VR se podrá añadir al camino prefijo Path, si hay un set S en ER que conecta R_k con uno de los tipos de registro de Path, y una de las condiciones siguientes es cierta.

1.- R_k es registro miembro de S y la raíz de Path es el registro propietario de S.

2.- R_k es registro propietario de S y por lo menos uno de los atributos de X (que no aparece en Path), aparece en R_k o en algún ancestro de R_k que no está en Path.

Si todo atributo de X aparece en el camino de acceso P y el coste de P es menor que el coste óptimo actual, entonces se actualiza el coste óptimo y P es el camino de acceso óptimo actual. Si se puede añadir un tipo de registro de VR al camino prefijo como el propietario de más de un enlace en EA, elegimos aquel que tenga un coste menor. Habremos encontrado el camino de acceso óptimo, cuando el paso iterativo para cada paso inicial haya terminado.

Ejemplo 4.6

Sea el esquema de red de la figura 2.4 y el query siguiente:

DISPLAY S_NOM, P#, CANT

Este query puede responderse calculando el O-transversal. El O-grafo contiene los seis tipos de registro del esquema de red.

La figura 4.9 muestra la porción del árbol de decisión que es recorrida durante la ejecución de OPT-B.

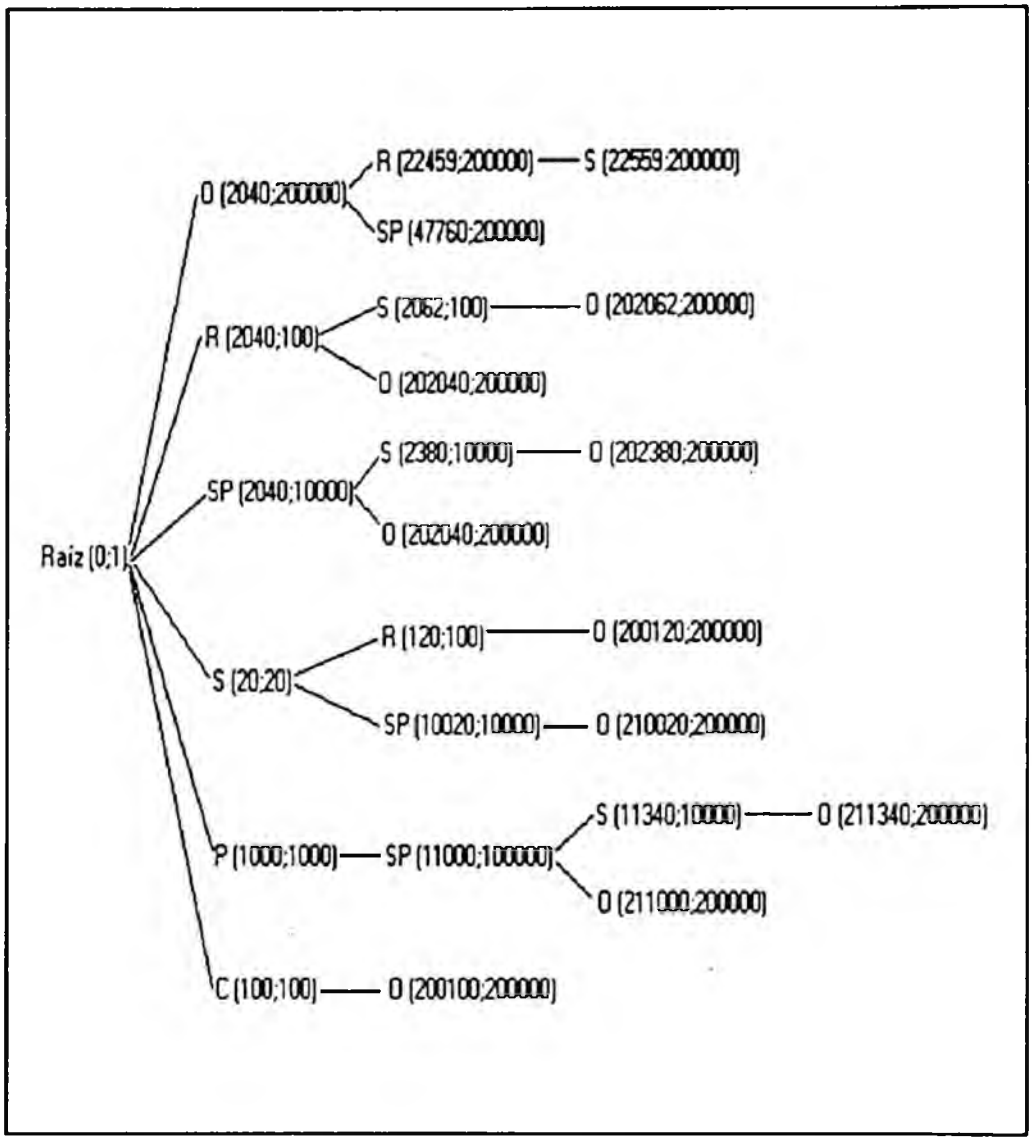


Figura 4.9

Arbol de decisión para la ejecución de OPT-B.

Si el registro O es el de cabecera, sólo se pueden añadir al camino prefijo los registros R y SP. Como registro cabecera, O, sólo puede ser recuperado mediante un recorrido secuencial de todas las áreas que contengan alguna ocurrencia de O. Si R es el siguiente registro a ser añadido al camino prefijo, el mejor medio para acceder a R es por el de clasificación de propietario con una relación. Si el registro S se recupera inmediatamente después de haber recuperado R, el coste será de 100 accesos a disco. El camino de acceso óptimo para este query será entonces O,R,S y el coste óptimo 22.559 accesos a disco.

4.3.3 OPTIMIZACION DE UN QUERY DADO POR UN PANEL DE UNA FILA

El apéndice C.7 muestra el algoritmo para procesar un query simple, Q, representado por un panel con una fila, y referido a un conjunto de atributos X.

El algoritmo PQ primeramente, enumera todos los R-transversales que deben ser evaluados para responder el query. El conjunto S de tipos de registros para los transversales, se inicializa como conjunto vacío. Todo registro tipo R_i del esquema de red, tal que $X \subseteq R_i$, se

incluye en dicho conjunto. A continuación, se excluye de él todo R_i que tenga un ancestro, R_j, en S. Finalmente, para cada R_i de S, se aplica el algoritmo de optimización OPT-A u OPT-B para encontrar un camino de acceso óptimo para el cálculo de R_i-transversal y se genera y ejecuta el código necesario para responder el query.

4.4 OPTIMIZACION DE UN QUERY GENERAL SPJ

Los queries que se expresan usando sólo operadores relacionales, SELECT, PROYECT, y JOIN, es decir, queries SPJ, pueden representarse mediante paneles con una o múltiples filas y un bloque de condiciones [AHOS79, KUCK82].

Dada una fila W de un panel, las tuplas recuperadas para ella tienen valores para todos los atributos A cuyas columnas en W tengan:

- una variable diferenciada,
- una variable no diferenciada que aparece en más de una fila o en el bloque de condiciones,
- una constante.

Por tanto, cada fila tiene un camino de acceso diferente para la recuperación de sus tuplas.

Entre cada par de filas W_i y W_j, hay dos tipos de interacciones posibles:

- Las filas W_i y W_j tienen una **interacción fuerte**, si aparece la variable v en ambas, o bien en el bloque de condiciones aparece una expresión lógica como "u=v", donde u es una variable de W_i y v de W_j.

- Las filas W_i y W_j tienen una **interacción débil**, si tienen una interacción fuerte, o bien hay una expresión lógica en el bloque de condiciones, tal que todas las variables de dicha expresión aparecen o en W_i o en W_j.

Dado un query consistente en las filas $W_1,\ldots,W_n$ podemos construir un grafo de interacción. Cada fila W_i ($1 \le i \le n$) es un nudo W_i del grafo y por cada par de nudos W_i y W_j del grafo, creamos un enlace E_{ij}, si hay interacción fuerte o débil entre las filas que representan. Los grafos de interacción para un query son útiles para optimizar su proceso.

Cuando el panel de un query tiene más de una fila, recuperamos una tupla por cada una de ellas, y si se satisfacen las condiciones del bloque de condiciones,

imprimimos los valores de las variables de la fila resumen. Un query de este tipo se puede evaluar usando una aproximación relacional o en red.

4.4.1 APROXIMACION EN RED

Cuando el panel de un query tiene más de una fila, recuperamos todas las tuplas posibles para la primera fila. Por cada tupla recuperada para la primera fila, recuperamos todas las tuplas de la segunda, y así sucesivamente. En este caso, se guardan los punteros del registro en curso y los valores de los atributos correspondientes a las variables del panel que están en más de una fila o en una expresión lógica aún no evaluada.

Consideremos ahora el orden de las filas, y por tanto, el de recuperación de las tuplas. El coste de recuperación puede ser sustancialmente diferente al variar el orden de las filas.

Si el panel de un query tiene n filas, entonces hay n! permutaciones diferentes. Para reducir el número de permutaciones usamos la heurística siguiente: sólo consideraremos aquellas ordenaciones $W_{i1},\ldots,W_{in}$ en que para todo j (j=2,...,n),

- hay por lo menos una interacción (fuerte o débil) entre W_{ij} y alguna W_{ik}, donde k<j, o bien
- para todo k>j no hay interacción entre W_{ij} y cualquiera de las $W_{i1},\ldots,W_{ij-1}$

Para encontrar la ordenación óptima de filas para la recuperación de todas las tuplas de cada fila, construimos un grafo de decisión de todas los posibles ordenaciones, como sigue.

Empezamos con el nudo raíz. Por cada fila del panel se crea un nudo y un enlace con la raíz. Cada nudo consiste en un conjunto de filas y una ordenación con su coste y tamaño. Extendemos el grafo de decisión, añadiendo en cada paso, una fila a la ordenación dada en cada nudo, hasta que todas las filas del panel estén en ella. Esto lo haremos utilizando una de las estrategias siguientes.

Sea $S=\{W_{i1},\ldots W_{ij}\}$ el conjunto de filas de un nudo.

1.- Si hay una interacción fuerte entre alguna W_{ik} ($1\leq k\leq j$) y algunas filas que no están en S, sólo aquellas filas que tengan una interacción fuerte con alguna W_{ik} ($1\leq k\leq j$) pueden añadirse a la ordenación dada.

2.- Si hay una interacción débil entre alguna W_{ik} ($1\leq k\leq j$) y algunas filas que no están en S, sólo

pueden añadirse aquellas filas que tengan una interacción débil con alguna W_{ik} ($1 \leq k \leq j$).

3.- Si no hay interacción (fuerte o débil) entre alguna fila de S y otras no presentes en S, entonces estas pueden ser añadidas.

Para cada elemento de una fila W_{ij} ($j=2,\ldots,n$), que sea una variable, si dicha variable aparece también en alguna fila W_{ik} donde $k<j$, la reemplazamos por el valor que acaba de recuperarse para ella en el bucle correspondiente a la fila W_{ik} [KUCK82].

Supongamos que v es una variable que aparece en la fila W_{ij} ($j=2,\ldots,n$); u es una variable que aparece en una fila W_{ik} ($k<j$) y en el bloque de condiciones aparece u=v, en este caso, reemplazamos v por el valor recuperado para u en el bucle correspondiente a W_{ik}.

Ejemplo 4.7

Sea el esquema de una base de datos de red, de la figura 4.6 y el query:

```
DISPLAY u.A, y.D, x.K
WHERE u.C=v.C AND v.D=<constante> AND v.I>y.K
AND w.D=x.H AND w.K<z.K AND x.K=z.P
```

En la figura 4.10 aparece el panel que lo representa.

Entre las filas 1 y 2 hay una interacción fuerte. Por tanto, cuando una de ellas pertenece a una ordenación, sólo puede añadirse a ella, la otra fila.

Entre la 5 y la 2 hay una interacción débil, y ésta es la única interacción de la fila 5 con cualquier otra. Por tanto, cuando la fila 5 está ya en la ordenación, sólo puede ser añadida a la misma, la fila 2; y cuando la ordenación es 1-2 o 2-1, sólo puede añadirse la fila 5.

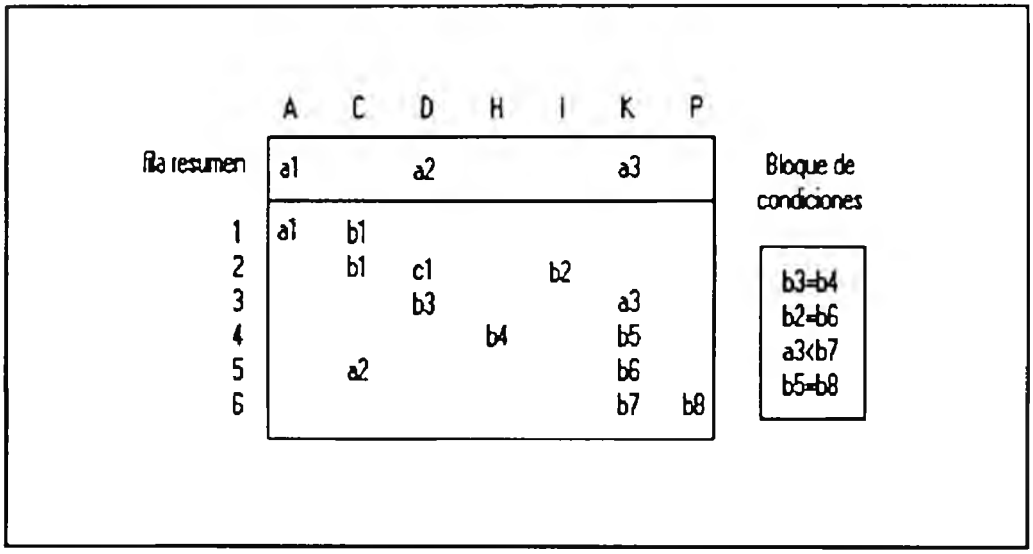


Figura 4.10

Panel del query del ejemplo 4.7.

En la figura 4.11, se muestran los grafos de interacción fuerte y débil.

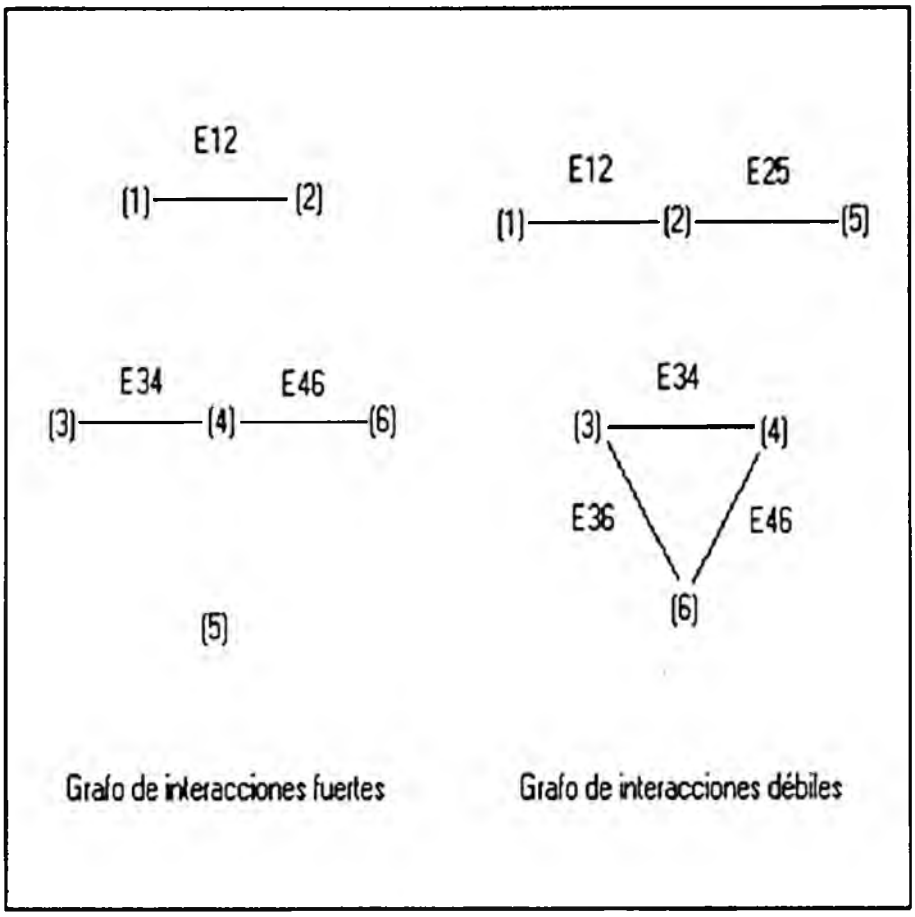


Figura 4.11

Grafos de interacciones para el query.

El grafo de decisión que enumera todas las posibles ordenaciones de filas para la evaluación de un query dado se muestra en la figura 4.12.

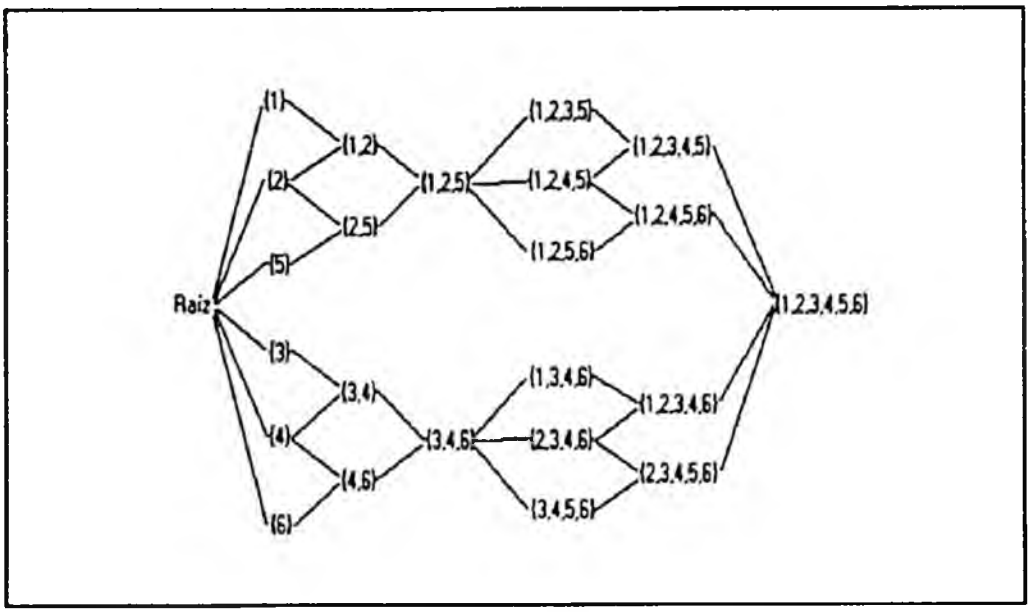


Figura 4.12

Grafo de decisión para evaluar un query dado.

4.4.2 APROXIMACION RELACIONAL

Mediante una aproximación relacional, evaluamos cada fila W_i del panel y guardamos el resultado en una relación intermedia cuyo esquema de relación es U_i. Esta relación contiene un atributo por:

- cada variable diferenciada de W_i,
- cada variable no diferenciada de W_i que está presente en alguna otra fila del panel del query,

- cada variable no diferenciada v de W_i que aparece en alguna expresión lógica del bloque de condiciones, de forma que dicha expresión tiene variables que aparecen en dos o más filas del panel del query.

Todas las expresiones lógicas del bloque de condiciones, con variables que aparecen en W_i, sólo se aplican al evaluar dicha fila.

Sean $W_1,..,W_n$ las filas del panel de un query y U_i el esquema de relación del resultado intermedio de la evaluación de la fila W_i. Para calcular el resultado de query, pueden utilizarse dos tipos de operaciones:

- Si hay una interacción fuerte entre W_i y W_j entonces, U_i y U_j son combinadas por una join natural o 'equi-join'.
- En caso contrario, se aplica un producto cartesiano de U_i y U_j (U_i x U_j).

El grafo de interacción fuerte, para un panel de múltiples filas, consta de uno o más subgrafos. Para cada subgrafo conectado, se elige un orden óptimo de combinación de las relaciones intermedias correspondientes a las filas del mismo. Para elegir el orden óptimo de unión de relaciones intermedias, enumeramos primeramente todos las ordenaciones de filas posibles, como sigue:

Toda fila del subgrafo puede ser la primera en alguna ordenación de filas.

A cada ordenación $W_{i1},\ldots,W_{ij}$, podemos añadir una fila W_k del subgrafo si hay un enlace entre W_k y alguna W_{im} ($1 \leq m \leq j$) de dicho subgrafo.

Por cada ordenación $W_{i1},\ldots,W_{in}$ del subgrafo conectado, calculamos el coste de evaluar U_{i1} *...* U_{in}, eligiendo finalmente, la ordenación de filas más económica.

Entre los diferentes métodos que existen, de implementación de la operación JOIN, destacamos los tres siguientes:

- Método de bucle anidado y búsquedas intercaladas [SELI79].
- Método de bloque anidado [KIM82] que generalmente es óptimo o casi óptimo.
- Método de 'Hybrid-hash join' [DEWI84], que es óptimo si se dispone de un buffer con gran capacidad.

Supongamos que un query dado Q, tiene un grafo de interacción, que posee los subgrafos $G_1,\ldots,G_h$ conectados. Para cada subgrafo G_i, hallamos un orden óptimo para realizar las operaciones 'join' y el resultado lo guardamos en una relación intermedia cuyo esquema es ES_i.

Para evaluar el query, hallamos el orden a seguir en el producto cartesiano ES_{i1} x···x ES_{ih} como sigue. El primer esquema de relación de esta ordenación puede ser cualquier S_i. A cada ordenación ES_{i1} x···x ES_{ij} ya fijada, añadimos un esquema de relación ES_k, si hay una interacción débil entre alguna fila de $G_{i1},\ldots,G_{ij}$ y alguna de G_k, o si no hay interacción entre alguna fila de $G_{i1},\ldots,G_{ij}$, y alguna que no está en $G_{i1},\ldots,G_{ij}$.

La filosofía para utilizar este tipo de heurísticas, consiste en aplicar las selecciones lo antes posible.

Ejemplo 4.8

Considerese el esquema de bases de datos en red de la figura 4.6 y el query del ejemplo 4.7.

Según el grafo de interacción fuerte de la figura 4.11 hay 3 subgrafos conectados. El subgrafo G1 consta de las filas 1 y 2 y el enlace E12. El G2 contiene las filas 3, 4 y 6 y los enlaces E34 y E46. El subgrafo G3 contiene la fila 5.

G1 puede evaluarse uniendo las filas 1 y 2 en orden 1-2 o 2-1. G2 puede evaluarse uniendo las filas 3, 4 y 6 en una de los ordenaciones siguientes: 3-4-6, 4-3-6, 4-6-3, o 6-4-3. La figura 4.13 muestra

todos las ordenaciones posibles para el producto cartesiano de las relaciones de resultados intermedios ES1, ES2 y ES3. Entre la fila 2 de G1 y la 5 de G3, hay una interacción débil. Por tanto, no son aplicables las ordenaciones ES1 x ES2 x ES3, y ES3 x ES2 x ES1.

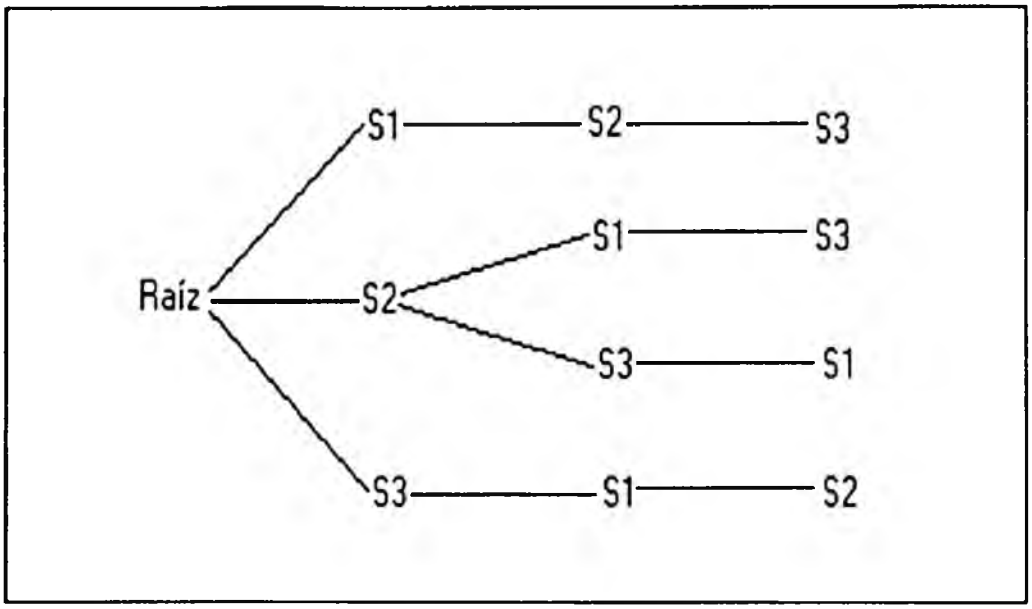


Figura 4.13

Ordenaciones posibles para el producto cartesiano.

4.4.3 CONSIDERACIONES SOBRE LAS APROXIMACIONES

Dado el panel de un query, consistente en las filas $W_1,\ldots,W_n$ podemos usar la aproximación en red o la relacional, ya discutidas, para evaluar el query. Ninguna de las dos aproximaciones es la mejor para su evaluación completa. Las filas de algunos subgrafos conectados pueden evaluarse mejor usando la aproximación en red, mientras que las filas de otros pueden evaluarse mejor con una aproximación relacional. Por tanto, lo que podemos hacer es combinar ambas aproximaciones.

Para cada subgrafo conectado, del grafo de interacciones fuertes de un query, consideramos ambas aproximaciones, y elegimos la que tenga un costo menor. Finalmente, evaluamos el producto cartesiano de las relaciones intermedias de cada subgrafo conectado, usando la aproximación relacional.

Ejemplo 4.9

Sea la figura 4.6, el esquema de una base de datos en red y el query del ejemplo 4.7.

Supongamos que las filas 1 y 2 como grupo, pueden evaluarse más eficientemente mediante una aproximación en red, mientras que las filas 3, 4 y 6 como grupo, con una aproximación relacional.

Los resultados de evaluar las filas 1 y 2 y de las filas 3, 4 y 6 se almacenan en las relaciones intermedias correspondientes a los esquemas ES1 y ES2, respectivamente.

La fila 5 no tiene interacción fuerte con ninguna otra fila del panel del query, y por tanto, se evalúa por separado, guardándose el resultado en la relación intermedia de esquema ES3. Finalmente, se aplican las ordenaciones de productos cartesianos enumerados en la figura 4.12 para localizar así un orden óptimo.

CAPITULO 5

CONCLUSIONES

5.1 ANALISIS DE LAS APORTACIONES ORIGINALES DE LA INVESTIGACION

La adopción, para un sistema de información, del enfoque de bases de datos persigue una serie de objetivos: independencia de datos, no redundancia, capacidad de representación, integridad, seguridad y confidencialidad de la información, etc. Entre estos objetivos básicos, destacamos los siguientes:

- Eficiencia referida a la utilización del espacio
- Disponibilidad: referida al tiempo de respuesta óptimo
- Accesibilidad: referida a la facilidad de acceso y manejo, de cara al usuario.

El uso de punteros en los sistemas de bases de datos en red, asegura la utilización del espacio de memoria eficientemente, reduciendo el número de datos redundantes.

En esta investigación se ha perseguido el objetivo de mejorar el rendimiento en la recuperación de información de las bases de datos en red.

Para conseguir este objetivo hemos desarrollado dos nuevos métodos, y para ello hemos tenido en cuenta que el número total de accesos a disco para procesar un query es clave en el rendimiento de las bases de datos. Estos métodos son aplicables tanto independientemente como en combinación con los métodos tradicionales. El módulo optimizador para la evaluación de queries sobre sistemas de bases de datos en red, presentado, está compuesto por una serie de algoritmos que aplican dichos métodos.

Para estudiar el rendimiento de estos métodos originales, hemos utilizado, como modelo de coste, la versión corregida del algoritmo desarrollado por Dayal y Goodman, para el cálculo del número de accesos a páginas de disco en la evualación de queries de árbol, para una base de datos en red.

Para la evaluación de un query sobre una base de datos en red se puede optar por:

- seguir los punteros entre las ocurrencias de registros, o bien
- guardarlos para utilizarlos posteriormente.

En caso de no realizar una de estas operaciones, se pierde la conexión entre los registros de un misma tupla, y es imposible obtener la tupla resultado.

Las investigaciones realizadas hasta la fecha sobre la evaluación de queries contra bases de datos en red, llevan a cabo la recuperación por tuplas siguiendo los punteros.

En esta investigación hemos demostrado que no es necesario seguir los punteros en todos los casos y que de llevarse a cabo no se obtiene un número menor de accesos a páginas de disco.

Para la recuperación de información de bases de datos en red, proponemos dos nuevos métodos, que en lugar de seguir los punteros, los almacenan temporalmente en una relación intermedia de base de datos. Dichos métodos combinan los métodos tradicionales, en red y relacionales.

Al aplicar nuestros métodos de recuperación, hemos mejorado el rendimiento sustancialmente en la evaluación de queries para bases de datos en red.

Además, dichos métodos pueden ser utilizados tanto para bases de datos en red, tipo CODASYL, como para las relacionales implementadas con un modelo en red o con punteros, y en combinación con otras formas de optimización existentes.

Un sistema de gestión de bases de datos en red, sin un interfaz con un lenguaje de interrogación de alto nivel, presentará para el usuario un grado de dificultad elevado.

En los sistemas de gestión de bases de datos navegacionales, dicho interfaz permite al usuario ignorar la organización física de la base de datos y los caminos de acceso disponibles. De esta forma, el usuario sólo tiene que especificar la información deseada y el interfaz se encargará de traducir esta petición en el correspondiente query de red, que se obtenga el mejor tiempo de respuesta. En el proceso de traducción, aparece el problema de la optimización de queries, y más en concreto, el de la localización de un camino de acceso óptimo en la base de datos.

Para resolver este problema de optimización Kuck, Sagiv e Illarramendi utilizan un algoritmo heurístico y Dayal y Goodman muestran el mejor modo de responder a un query de árbol para bases de datos en red. Estas aproximaciones asumen que los punteros deben seguirse.

Además, al desarrollar los algoritmos de optimización, hemos tenido en cuenta el hecho de que un query no procedural puede responderse correctamente, evaluando uno de los diferentes árboles de acceso posibles; en contraposición a las investigaciones anteriormente citadas, que sólo consideran un árbol de acceso para encontrar un camino de acceso óptimo.

Nuestros algoritmos están desarrollados para un sistema en red con un interfaz que proporciona un lenguaje de interrogación tipo SQL.

Los algoritmos de optimización de queries que presentamos están desarrollados por pasos, comenzando por la localización de un camino de acceso óptimo para un árbol de acceso dado, aplicando tanto el método tradicional como nuestros propios métodos.

La optimización del camino de acceso para la evaluación de un R-transversal se realiza enumerando en primer lugar todos los árboles de acceso posibles para el R-transversal y localizando, a continuación, un camino de acceso óptimo para cada uno.

Para un query general SPJ representado por un panel con múltiples filas y un bloque de condiciones, proponemos tanto la aproximación relacional como la de red con iteración anidada.

Dado que hemos constatado que ninguna de estas aproximaciones, la de red y la relacional, es el medio óptimo para evaluar algunos queries, proponemos otro método que combina ambas aproximaciones.

Los métodos propuestos, pueden extenderse fácilmente para la optimización de la evaluación de queries contra bases de datos relacionales implementadas con punteros [GIL90].

5.2 LINEAS FUTURAS DE INTERES

Uno de los campos que no hemos investigado es el efecto de la memoria virtual sobre los algoritmos de proceso de queries y su optimización para bases de datos relacionales con punteros. En los gráficos de la figura 3.11, se muestra que cuando el número de páginas de memoria disponible para los buffers aumenta, el coste de clasificación y de combinación de las relaciones se reduce, y como consecuencia el método de clasificación de propietario es más factible. En el mundo real, al trabajar con memoria virtual se proporciona a cada usuario un espacio de memoria amplio; sin embargo, sólo una pequeña parte reside en le memoria principal.

El problema teórico resuelto en esta investigación, junto con el de la generación automática del algoritmo que permita ejecutar el plan seleccionado, es paralelo al problema estudiado en inteligencia artificial, del paso automático de una especificación al algoritmo correspondiente, por lo que una de las soluciones factible sería desarrollar un sistema experto que realizase el proceso de optimización.

Otra linea de acción es la caracterización de los queries. Es decir, el estudio de los queries para su agrupamiento en tipos generales, para que se pueda aplicar, de los métodos de acceso propuestos, el más apropiado a cada uno de los tipos obtenidos, con la consiguiente mejora del rendimiento del optimizador.

BIBLIOGRAFIA

[AHOB79] A. V. Aho, C. Berri, J. D. Ullman
The Theory of Joins in Relational Databases
ACM Transac. on Database Systems, Vol 4, nº3, 1979

[AHOS79] A. V. Aho, Y. Sagiv, J. D. Ullman
Efficient Optimization of a Class of Relational Expressions
ACM Transac. on Database Systems, Vol 4, nº4, 1979

[ASTR80] M. M. Astrhan, W. Kim, M. Schokolnick
Evaluation of the System R Access Path Selection Mechanism
Proc. IFIP Congress 1980

[BEER79] C. Beeri, P. A. Bernstein
Computational Problems Related to the Design of Normal Form Relational Schemas
ACM Transac. on Database Systems, Vol 4, nº1, 1979

[BERN76] P. A. Bernstein
Synthesizing Third Normal Form Relations from Funtional Dependencies
ACM Transac. on Database Systems, Vol 1, nº4, 1976

[BERN81] P. A. Bernstein, N. Goodman, E. Wong et al.
Query processing in a System for Distributed Databases (SDD-1)
ACM Transac. on Database Systems, Vol 6, nº4, 1981

[BERT85] M. Bert, G. Ciardo, M. L. Demarie et al.
A Relational Interface for CODASYL Databases
Conferencia Informática Latina CIL 1985

[BING79] S. Bing Yao
Optimization of Query Evaluation Algorithms
ACM Transac. on database Systems, Vol 4, nº2, 1979

[CHOL86] L. Cholvy, R. Demolombe
Querying a Rule Base
First Conf. on Expert Database Systems, 1986

[CODA78] CODASYL Data Description Language. Journal of Development
Imforation Systems, Vol 3, nº4, 1978

[CODD70] E. F. Codd
A Relational Model of Data for Large Shared Data Banks
Communications of ACM, June 1970

[DATE86] C. J. Date
An Introduction to Database Systems
Addison Wesley, Fourth Edition, 1986

[DAYA82] U. Dayal, N. Goodman
Query Optimization for Codasyl Database Systems
Proc. ACM SIGMOD Int. Conf. on Management of Data, 1982

[DEMO80] R. Demolombe
Estimation of the Number of Tuples Satiafying a Query Expressed in Predicate Calculus Language
Proc. VLDB, 1980

[DEMO83] R. Demolombe
How to Improve the Performance of Relational DBMS
Proc. IFIP, 1983

[DEWI84] D. J. DeWitt, R. H. Katz, F. Olken et al.
Implementation Techniques for Main Memory Database Systems
Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984

[FAGI82] R. Fagin, A. O. Mendelzon, J. D. Ullman
A Simplified Universal Relation Assumption and Its Properties
ACM Transac. on database Systems, Vol 7, nº3, 1982

[GALL81] J. Gallier
Advances in Data Base Theory - Vol 1
Plenum Press, 1981

[GIL90] M. J. Gil
Comparing the Performance of the Different Database Systems Running on DEC
C.A. Computer Associates U.K., 1990

[GRAY82] P. M. D. Gray
Use of Automatic Programming and Simulation to Facilitate Operations on CODASYL Databases
In State of the Art Report Database, Series 9, nº8 1982

[ILLA87] A. Illarramendi
Formalización, Diseño e Implementación de una Interfaz Relacional Eficaz para Sistemas CODASYL
Tesis Doctoral, Facultad de Informática UPV, 1987

[KATZ82] R. H. Katz
Compilation of Relational Queries into CODASYL DML
Improving Database Usability, Academic Press Inc. 1982

[KIN82] W. Kim
On Optimizing an SQL-Like Nested Query
ACM Transac. on Database Systems, Vol7, nº3, 1982

[KIMR85] W. Kim, S. Reiner, S. Batory
Query Porcessing in Database Systems
Springer-Verlag, Berlín 1985

[KUCK82] S. M. Kuck, Y. Sagiv
An Universal Relation Database System Implemented Via the Network Model
Proc. ACM Symposium on Principles of Database Systems 1982

[KUCK83] S. M. Kuck, Y. Sagiv
Designing Globally Consistent Network Schemas
Proc. ACM SIGMOD Int. Conf. on Management of Data, 1983

[KUCK84] S. M. Kuck, Y. Sagiv
Links in Relational Databses
Proc. CIPS, Canada, 1984

[LIEN82] Y. E. Lien
On the Equivalence of Database Models
Journal of ACM, Vol 29, nº2, 1982

[PETE81] J. S. Peterson
Query Time Calculations in Networks
M.S. Thesis, Dep. Of Computer Science, Illinois 1981

[REIN77] E. M. Reingold, J. Nievergelt, N.Deo
Combinatorial Algorithms, Theory and Practice
Prentice-Hall 1977

[REIN85] S. Reiner, A Rosenthal
Querying Relational Views of Networks
Query Processing in Database Systems Spring-Verlag 1985

[RICH81] P. Richard
Evaluation of the Size of a Query Expressed in Relational Algebra
Conf. ACM SIGMOD, 1981

[SALT88] F. Saltor
Bases de Datos
EUSTAT cuaderno 13, 1988

[SELI79] P.G. Selinger, M.M. Astrahan, R.A. Lorie, T.G. Price
Access Path Selection in a Relational Database System
Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979

[ULLM82] J.D. Ullman
Principles of Database Systems
Computer Science Press, Second edition, 1982

[WONG76] E. Wong, and K.Youssefi
Decomposition: A Strategy for Query Processing
ACM Transac. on Database Systems, Vol 1, nº3, 1976

[YAO77] S. B. Yao
Approximating Block Accesses in Database Organizations
Communications of ACM, Vol 20, nº4, 1977

[YAO79] S. B. Yao
Optimization of Query Evaluation Algorithms
ACM Transac. on Database Systems, Vol 4, nº2, 1979

[ZANI79] C. Zaniolo
Design of Relational Views over Network Schemas
Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979

APENDICE A

ESQUEMA DE LA BASE DE DATOS EJEMPLO

```
SCHEMA NAME IS EJTESIS
*
*-------------------------------- Definición de áreas
*
AREA NAME IS A1
*
*-------------------------------- Definición de registros
*
RECORD NAME IS S
     WITHIN A1
          ITEM NAME IS S#
               TYPE IS UNSIGNED NUMERIC 4
          ITEM NAME IS S_NOM
               TYPE IS CHARACTER 20
          ITEM NAME IS S_CIUDAD
               TYPE IS CHARACTER 20
*
RECORD NAME IS P
     WITHIN A1
          ITEM NAME IS P#
               TYPE IS UNSIGNED NUMERIC 4
          ITEM NAME IS P_NOM
               TYPE IS CHARACTER 20
          ITEM NAME IS COLOR
               TYPE IS CHARACTER 4
*
RECORD NAME IS C
     WITHIN A1
          ITEM NAME IS C#
               TYPE IS UNSIGNED NUMERIC 4
          ITEM NAME IS C_NOM
               TYPE IS CHARACTER 20
          ITEM NAME IS C_CIUDAD
               TYPE IS CHARACTER 20
*
RECORD NAME IS R
     WITHIN A1
          ITEM NAME IS R#
               TYPE IS UNSIGNED NUMERIC 4
          ITEM NAME IS R_NOM
               TYPE IS CHARACTER 20
          ITEM NAME IS SALARIO
               TYPE IS CHARACTER 4
*
RECORD NAME IS SP
     WITHIN A1
          ITEM NAME IS PRECIO_U
               TYPE IS UNSIGNED NUMERIC 4
*
RECORD NAME IS O
     WITHIN A1
          ITEM NAME IS CANT
               TYPE IS UNSIGNED NUMERIC 4
*
*
```

```
*-------------------------------- Definición de registros
*
SET NAME IS ALL_S
     OWNER IS SYSTEM
     MEMBER IS S
*
SET NAME IS ALL_P
     OWNER IS SYSTEM
     MEMBER IS P
*
SET NAME IS ALL_C
     OWNER IS SYSTEM
     MEMBER IS C
*
SET NAME IS S_R
     OWNER IS S
     MEMBER IS R
*
SET NAME IS S_SP
     OWNER IS S
     MEMBER IS SP
*
SET NAME IS P_SP
     OWNER IS P
     MEMBER IS SP
*
SET NAME IS R_O
     OWNER IS R
     MEMBER IS O
*
SET NAME IS SP_O
     OWNER IS SP
     MEMBER IS O
*
SET NAME IS C_O
     OWNER IS C
     MEMBER IS O
```

APENDICE B

RESULTADOS DEL RENDIMIENTO DE LOS METODOS DE RECUPERACION

B.1 METODO DE CLASIFICACION DE PROPIETARIO CON UN MIEMBRO POR SET

TUPLAS/BLOQUE	INCREMENTO VELOCIDAD
2	0.15
4	0.25
5	0.3
12	0.4
15	0.47
16	0.5
20	0.55
24	0.6
25	0.62
26	0.66
35	0.7
40	0.75
45	0.76
46	0.77
52	0.77
53	0.78
58	0.78
60	0.83
69	0.85
70	0.87
79	0.87
80	0.88
88	0.88
90	0.89
94	0.89
98	0.9
102	0.9

B.2 METODO DE CLASIFICACION DE PROPIETARIO CON DOS MIEMBROS POR SET

TUPLAS/BLOQUE	INCREMENTO VELOCIDAD
4	0.25
5	0.32
10	0.43
11	0.55
12	0.7
16	0.74
20	0.8
25	0.89
28	1.0
49	1.7
50	1.27
52	1.3
60	1.35
61	1.39
62	1.4
64	1.42
80	1.5
81	1.56
82	1.56
83	1.57
88	1.58
99	1.58
100	1.55
102	1.6

B.3 METODO DE CLASIFICACION DE PROPIETARIO CON CUATRO MIEMBROS POR SET

TUPLAS/BLOQUE	INCREMENTO VELOCIDAD
5	0.25
10	0.5
12	0.7
24	1.15
25	1.25
46	1.8
50	1.95
52	1.98
60	2.1
65	2.2
72	2.25
80	2.37
85	2.43
100	2.58
102	2.65
103	2.68

B.4 METODO DE CLASIFICACION DE PROPIETARIO CON DIECIESEIS MIEMBROS POR SET

TUPLAS/BLOQUE	INCREMENTO VELOCIDAD
5	0.25
9	0.5
12	0.84
16	1.0
23	1.45
28	1.6
49	2.8
50	3.05
78	4.2

B.5 PUNTOS DE RUPTURA Y FACTORES DE BLOQUEO PARA UN BUFFER DE CUATRO PAGINAS

	PUNTOS DE RUPTURA		
OCUR. MIEMB.(x1000)	2 MIEMB/SET	4 MIEMB/SET	64 MIEMB/SET
1	17	11	9
2	16	13	11
4	20	14	11
8	20	16	13
16	25	16	13
32	25	19	15
64	29	19	15
128	30	22	17
256	32	22	19
512	37	25	19
1024	37	27	21
2048	40	27	21
4096	40	30	23
8192	45	30	24
16284	46	32	25
32568	49	32	25
65536	52	35	27

B.6 PUNTOS DE RUPTURA Y FACTORES DE BLOQUEO PARA UN BUFFER DE OCHO PAGINAS

	PUNTOS DE RUPTURA		
OCUR. MIEMB.(x1000)	2 MIEMB/SET	4 MIEMB/SET	64 MIEMB/SET
1	11	8	7
2	13	8	7
4	13	8	7
8	13	11	9
16	16	11	9
32	16	11	9
64	16	14	11
128	20	14	11
256	20	14	11
512	20	16	13
1024	25	16	13
2048	25	16	13
4096	25	18	15
8192	29	19	15
16284	29	19	15
32568	29	20	17
65536	32	22	17

B.7 PUNTOS DE RUPTURA PARA DOS OCURRENCIAS MIEMBRO POR SET Y BUFFERS DE TAMAÑO DIFERENTE

	BUFFERS		
OCUR. MIEMB.(x1000)	4 PAGINAS	8 PAGINAS	32 PAGINAS
1	17	11	8
2	16	13	8
4	20	13	8
8	20	13	8
16	25	16	9
32	25	16	13
64	29	16	13
128	30	20	13
256	32	20	13
512	37	20	13
1024	37	25	16
2048	40	25	16
4096	40	25	16
8192	45	29	16
16284	46	29	16
32568	48	29	17
65536	52	32	20

B.8 PUNTOS DE RUPTURA PARA CUATRO OCURRENCIAS MIEMBRO POR SET Y BUFFERS DE TAMAÑO DIFERENTE

	BUFFERS		
OCUR. MIEMB.(x1000)	4 PAGINAS	8 PAGINAS	32 PAGINAS
1	11	8	6
2	12	8	6
4	13	8	6
8	16	11	6
16	16	11	8
32	19	11	8
64	19	14	8
128	22	14	8
256	22	14	8
512	25	16	9
1024	27	16	10
2048	27	16	10
4096	30	18	10
8192	30	19	10
16284	32	19	10
32568	32	20	14
65536	35	22	14

APENDICE C

ALGORITMOS DE OPTIMIZACION

C.1 ALGORITMO DE OPTIMIZACION POR TUPLAS: ALG1

Entrada: Un árbol de acceso T.

Salida: El camino de acceso óptimo Opath, coste Opcost y tamaño resultante Tamaño.

Paso iterativo: ITERA

```
procedure ALG1(T,Opath,Opcost,Tamaño)

begin
    Opcost <- ∞
    for cada tipo de registro R de T do  [1]
       C <- Σ dA  [2]
            A
       P <- (R, todas las áreas)
       if R puede ser accedido como miembro del set singular
         then
                  | Y(nR,NR,σR x NR)+dI [3]
                  | σR x nR + dI        [4]
            dR <- | NR                  [5]
                  | nR                  [6]

           if dR < C
             then
                 C <- dR
                 P <- (R, set singular)
           endif
         else
           if R es accesible por un camino dirigido R1,...,Rk,R
             then
               Calcular el coste CR para evaluar:
               (R1, set singular)‖...‖(R, miembro del set Sk)
               if CR < C
                 then
                   P<-(R1,set singular)‖...‖(R,miembro set Sk)
                   C <- CR
               endif
           endif
       endif
       if R puede ser accedido por una clave de cálculo
         then
           dR <- dH + dkey
           if dR < C
             then
               C <- dR
               P <- (R, clave de cálculo)
           endif
       endif
       if C < Opcost [7]
         then
           Tam <- σR x n
           if T contiene un solo registro
             then [8]
               Opcost <- C
               Opath <- P
               Tamaño <- Tam
             else [9]
               call ITERA (T, P, C, Tamaño, Tam)
           endif
       endif
    endfor
end
```

```
procedure ITERA (T, Path, Cost, Tamaño, Ctamaño)

begin

    for cada tipo de registro R de T adyacente a Path do  [1]
       if R puede recuperarse como miembro de un set S
           then
                        | Y(nM,NM,σR x NM) + dI [3]
                        | σR x nM + dI         [4]
              C  <-     | NM                    [5]
                        | nM                    [6]

              C <- C x Ctamaño
              Tam <- Ctamaño x nM x σR
              M <- Miembro del set S
           else [10]

                        | σs                 [11]
              C <-      | σs/2 x (Nm + 1)    [12]
                        | σs/2 x (nM + 1)    [6]

              C <- C x Ctamaño
              Tam <- Ctamaño X σs x σR
              M <- Propietario del set S
       endif
       C <- Cost + C
       if C < Opcost [7]
         then
           Añadir (R,M) to Path [13]
           if todo registro de T está en P
             then [8]
               Opcost <- C
               Opath <- P
               Tamaño <- Tam
             else  [9]
                call ITERA (T, P, C, Tamaño, Tam)
           endif
       endif
    endfor
end
```

Comentarios del algoritmo

[1]: Para todos los métodos posibles M para recuperar R, calcular el coste y encontrar el método más económico

[2]: Búsqueda secuencial en todas las áreas

[3]: Si se usa un índice de agrupamiento

[4]: Si se usa un índice de no agrupamiento

[5]: Si hay agrupamiento pero no se usan índices

[6]: En cualquier otro caso

[7]: Branch and bound

[8]: Devolver camino óptimo y coste

[9]: Se necesita iteración

[10]: R se recupera como propietario del set S

[11]: Si se dispone de punteros owner

[12]: Si no hay punteros owner pero R está agrupado vía S

[13]: P<-Path(R,M)

C.2 ALGORITMO DE OPTIMIZACION: ALG2

Entrada: Un árbol de acceso T.

Salida: El camino de acceso óptimo **Opath**, coste Opcost y tamaño resultante **Tamaño**.

Paso iterativo: ITERA

```
procedure ITERA (T, Path, Cost, Tamaño, Ctamaño)

begin
    for cada tipo de registro R de T adyacente al Path do [1]
        if R puede recuperarse como miembro de un set S
          then
                 | Y(nM,NM,σR x NM)+dI [2]
                 | σR x nM + dI       [3]
            C <- | NM                  [4]
                 | nM                  [5]

            C <- C x Ctamaño
            Tam <- Ctamaño x nM x σR
            M <- Miembro del set S
          else  [6]

                 | σs                [7]
            C <- | σs/2 x(Nm + 1)     [8]
                 | σs/2 x (nM + 1)    [5]

            C <- C x Ctamaño
            Tam <- Ctamaño X σs x σR
            M <- Propietario del set S
            (* ------------------------------------ [9] *)
            Calcular el coste CR para este método
            if CR < C
              then
                C <- CR
                M <- Clasificación de propietario del set S
                     con una relación en serie
            endif
            (* ------------------------------------ [10] *)
            Calcular el coste CR para este método
            if CR < C
              then
                C <- CR
                M <- Clasificación de propietario del set S
                     con dos relaciones en serie
            endif
         endif
         C <- Cost + C
         if C < Opcost [11]
           then
             Añadir (R, M) to Path [12]
             if todo registro de T está en P
               then [13]
                 Opcost <- C
                 Opath <- P
                 Tamaño <- Tam
               else  [14]
                 call ITERA (T, P, C, Tamaño, Tam)
             endif
        endif
     endfor
end
```

Comentarios del algoritmo

[1]: Para todos los métodos posibles M para recuperar R, calcular el coste y encontrar el método más económico

[2]: Si se usa un índice de agrupamiento

[3]: Si se usa un índice de no agrupamiento

[4]: Si hay agrupamiento pero no se usan índices

[5]: En cualquier otro caso

[6]: R se recupera como propietario del set S

[7]: Si se dispone de punteros owner

[8]: Si no hay punteros owner pero R está agrupado vía S

[9]: Método de clasificación de propietario con una relación en serie

[10]: Método de clasificación de propietario con dos relaciones en serie

[11]: Branch and bound

[12]: P<-Path‖(R,M)

[13]: Devolver camino óptimo y coste

[14]: Se necesita más iteración

C.3 ALGORITMO DE OPTIMIZACION: ALG3

Entrada: Un árbol de acceso T con el registro R como raíz

Salida: Camino de acceso óptimo Opath; coste Opcost y tamaño del resultado Tamaño.

Paso down-top: UP

```
procedure ALG3 (T, Opath, Opcost, Tamaño)

begin
   call ALG2 (T, Opath, Opcost, Tamaño)
   if T contiene más de un tipo de registro
      then
        for cada subárbol T_i (1≤i≤m) con raíz Ro_i do
            call ALG3 (T_i, P_i, C_i, S_i)
        endfor
        Calcular el coste total CT  y el tamaño ST  de salvar,
        clasificar y combinar las m relaciones intermedias, y
        si es necesario recuperar R.
        if CT < Opcost
          then
            Opath <- Concatenación de los m P_i
            Opcost <- CT
        endif
        for cada k (1≤k<m) do
            for cada combinación de k subárboles diferentes de
                los m do
                Calcular  el coste total CT  y el tamaño ST de
                salvar, clasificar y combinar las k relaciones
                intermedias y recuperar R si es necesario
                if CT < Opcost
                  then
                    P <- Concatenación de los k P_i
                    Cop <- ∞
                    call UP(T, P, CT, ST, Pop, Cop)
                    if Cop < Opcost
                      then
                        Opath <- Pop
                        Opcost <- Cop
                    endif
                endif
            endfor
        endfor
    endif
end
```

```
procedure UP(T, Path, Cost, Tamaño, Opath, Opcost)

begin
    for cada registro R de T adyacente a Path do [2]

                | σs                [3]
          C <-  | σs/2 x (NM + 1)  [4]
                | σs/2 x (nM + 1)  [5]

          C <- C x Ctamaño
          Tam <- Ctamaño x σs x σR
          M <- Propietario del set S
          (* ------------------------------------------ [6] *)
          Calcular el coste CR para este método
          if CR < C
            then
             C <- CR
             M <- Clasificación de propietario del set S con una
                  relación en serie
          endif
          (* ------------------------------------------ [7] *)
          Calcular el coste CR para este método
          if CR < C
            then
             C <- CR
             M <- Clasificación de propietario del set S con dos
                  relaciones en serie
          endif
          C <- Cost + C
          if C < Opcost [8]
            then
              Añadir (R,M) to Path [9]
              if todos los registros de T están en P
                then [10]
                  Opcost <- C
                  Opath <- P
                else [11]
                  call UP(T, P, C, Tam, Opath, Opcost)
              endif
          endif
   endfor
end
```

Comentarios del algoritmo

[1]: Sea R un tipo de registro miembro de los sets $So_1,...,So_n$ y $Ro_1,...,Ro_n$, los propietarios de $So_1,...,So_n$ respectivamente

[2]: R sólo puede recuperarse como propietario del set S

[3]: Si hay punteros owner

[4]: Si no hay punteros owner pero R está agrupado vía S

[5]: En cualquier otro caso

[6]: Método de clasificación de propietario con una relación en serie

[7]: Método de clasificación de propietario con dos relaciones en serie

[8]: Branch and bound

[9]: P <- Path‖(R,M)

[10]: Devolver camino óptimo y coste

[11]: Se necesita más iteración

C.4 ALGORITMO DE CONSTRUCCION DE ARBOLES DE ACCESO: GRAFO

Entrada: Un esquema de red N, un tipo de registro R de N y un conjunto de atributos X.

Salida: El conjunto de todos los posibles árboles de acceso PT.

Paso iterativo: ARBOL

```
procedure GRAFO(N, R, X, PT)

begin
    VR <- {R}
    ER <- Ø
    for cada set Sj de N con el registro miembro R do
        ER <- ER U {Sj}
    endfor
    for cada Ri ancestro de R do
        VR <- VR U {Ri}
        for cada set Sj de N con el registro miembro Ri do
            ER <- ER U {Sj}
        endfor
    endfor
    PT <- Ø
    Vtemp <- {R}
    Etemp <- Ø
    if Vtemp = VR
      then
        PT <- {(VR, ER)}
      else
        call ARBOL(VR, ER, Vtemp, Etemp, PT)
   endif
end



procedure ARBOL(VR, ER, Vin, Ein, PT)

begin [1]
    Vtemp <- Vin U {Ri}
    if Vtemp = VR
      then
        for cada set Sj de ER poseido por Ri do
            ET <- Ein U {Sj}
            VT <- VR
            while exista un nudo hoja Rh en VT tal que
                  Rh ∩ X = Ø do
                  VT <- VT - {Rh}   [2]
                  ET <- ET - {Sm}
            endwhile
            for cada set singular Sj en ER con un
                registro  miembro Rj en VT do
                ET <- ET U {Sj}
            endfor
            PT <- PT U {(VT, ET)}
        endfor
      else
        for cada set Sj en ER poseido por Ri do
            Etemp <- Ein U {Sj}
            call ARBOL(VR, ER, Vtemp, Etemp, PT)
        endfor
    endif
end
```

Comentarios del algoritmo

[1]: Sea R_i un tipo de registro de VR que no está en Vin

[2]: Sea S_a el set poseído por R_h en ET

C.5 ALGORITMO DE OPTIMIZACION PARA UN R-TRANSVERSAL: OPT-A

Entrada: Un esquema de red N, un tipo de registro R de N y un conjunto de atributos X.

Salida: Un camino de acceso óptimo Opath

```
procedure OPT-A (N, R, X, Opath)

begin
    Opcost <- ∞  [1]
    call GRAFO (N, R, X, PT)
    for cada árbol de acceso T de PT do
        call ALG3(T, Path, Cost, Tamaño) [2]
        if Cost < Opcost
          then
            Opath <- Path
            Opcost <- Cost
        endif
    endfor
end
```

Comentarios del algoritmo

[1]: Enumerar todos los posibles árboles de acceso

[2]: O bien, call ALG2(T, Path, Cost, Tamaño)

C.6 ALGORITMO DE OPTIMIZACION PARA UN R-TRANSVERSAL: OPT-B

Entrada: Un esquema de red N, un tipo de registro R de N y un conjunto de atributos X.

Salida: Un camino de acceso óptimo Opath

Paso iterativo: ITERA-B

```
procedure  OPT-B(N, R, X, Opath)

begin [1]
    VR <- {R}
    ER <- Ø
    for cada set Sj de N con R como registro tipo miembro do
        ER <- ER U {Sj}
    endfor
    for cada ancestro Ri de R do
        VR <- VR U {Ri}
        for cada set Sj de N con Ri como miembro do
            ER <- ER U {Sj}
        endfor
    endfor
    (* ------------------------------------------------ [2] *)
    Opcost <- ∞
    for cada tipo de registro Ri de VR do [3],[4]
        C <- Σ dA [5]
             A
        P <- (Ri, todas las áreas)
        if  Ri  puede  ser  accedido  como  miembro  de  un set
            singular
          then

                    | Y(nR,NR,σRx NR) + dI [6]
              dR <- | σR x nR + dI        [7]
                    | NR                  [8]
                    | nR                  [9]

            if dR < C
              then
                C <- dR
                P <- (Ri, Set singular)
            endif
        endif
        if Ri puede ser accedido por una clave de cálculo
          then
            dR <- dH + dkey
            if dR < C
              then
                C <- dR
                P <- (Ri, Clave de cálculo)
            endif
        endif
```

```
        if C < Opcost [10]
          then
            Tam <- σR x nR
            if X ⊆ R̄i
              then [11]
                Opcost <- C
                Opath <- P
              else [12]
                Raíz <- Ri [13]
                W <- X - Ri [14]
                Zo <- Ø [15]
                Zm <- Ø
                for cada tipo de registro Rk, propietario de un
                    set con un tipo de registro
                    miembro Ri do
                    if W ∩ Rk = Ø
                      then [16]
                        Zo <- Zo U {Rk}
                    endif
                endfor
                for cada tipo de registro Rk miembro de  un set
                    poseido por Ri do
                    Zm <- Zm U {Rk} [17]
                endfor
                call ITERA-B(VR,ER,W,Zo,Zm,P,Raíz,C,Tam,Opath,
                             Opcost)
            endif
        endif
    endfor
end
```

```
procedure ITERA-B (VR,ER,X,Zo,Zm,Path,Raíz,Cost,Tamaño,Opath,
                   Opcost)
begin
    for cada tipo de registro R de (Zo U Zm) do [18]
        if R ∈ Zm [19]
          then

                    | Y(nM,NM,σR x NM)+dI [6]
              C <-  | σR x nM + dI        [7]
                    | NM                  [8]
                    | nM                  [9]

            C <- C x Ctamaño
            Tam <- Tamaño x nM x σR
            M <- Miembro del set S
          else [20]
            C <- ∞
            for cada set S_i de ER con R como propietario y R_m
                de Path, como miembro do

                      |1              [21]
                Cs <- |1/2x (NM+1)    [22]
                      |1/2 x (nM+1)   [9]

                if Cs < C
                  then
                    S <- S_i
                    C <- Cs
                    M <- Propietario del set S_i
                endif
            endfor
            C <- C x Tamaño
            Tam <- Tamaño x σR
           (* ---------------------------------------- [23] *)
            Calcular el coste CR para este método
            if CR < C
              then
                C <- Cr
                M <- Clasificación  de propietario del set S
                     con una relación en serie
            endif
           (* ---------------------------------------- [24] *)
            Calcular el coste CR para este método
            if CR < C
              then
                C <- CR
                M <- Clasificación de propietario del set S
                     con dos relaciones en serie
            endif
        endif
```

```
        C <- Cost + C
        if C < Opcost [10]
          then
            Añadir (R,M) to Path [25]
            W <- X - R
            if W = Ø
              then [11]
                Opcost <- C
                Opath <- P
              else  [12]
                if R ∈ Zm
                  then
                    Ym <- Ø
                    Nraíz <- R
                    for cada tipo de registro Rk miembro
                        de un set poseido por R do
                        Ym <- Ym U {Rk}
                    endfor
                  else
                    Ym <- Zm
                    Yo <- Zo - {R}
                endif
                for cada registro Rk propietario de
                    un set con R como miembro do
                    if W ∩ Rk NOT= Ø
                      then
                        Yo <- Yo U {Rk}
                      else
                        Yo <- Yo - {Rk}
                    endif
                endfor
                call ITERA-B (VR,ER,W,Yo,Ym,P,Nraíz,C,Tam,
                              Opath,Opcost)
            endif
        endif
    endfor
end
```

Comentarios del algoritmo

[1]: Construir R-grafo

[2]: Paso inicial

[3]: Registro de cabecera

[4]: Para todos los métodos posibles M para recuperar R_i, calcular el coste y encontrar el método más económico

[5]: Búsqueda secuencial en todas las áreas

[6]: Si se usa un índice de agrupamiento

[7]: Si se usa un índice de no agrupamiento

[8]: Si hay agrupamiento pero no se usan índices

[9]: En cualquier otro caso

[10]: Branch and bound

[11]: Devolver camino óptimo y coste

[12]: Se necesita iteración

[13]: Raíz del camino prefijo

[14]: Atributos aún no recuperados

[15]: Localizar registros accesibles

[16]: R_k se recuperará como propietario

[17]: R_k se recupera como registro miembro

[18]: Para todos los métodos posibles M para recuperar R, calcular el coste y encontrar el método más económico

[19]: R es localizado como un tipo de registro miembro de un set S

[20]: R se recuperará como propietario de un set S

[21]: Si existen punteros owner

[22]: Si no hay punteros owner

[23]: Método de clasificación de propietario con una relación en serie

[24]: Método de clasificación de propietario con dos relaciones en serie

[25]: P <- Path‖(R,M)

C.7 PROCESO DE UN QUERY REPRESENTADO POR UN PANEL: PQ

Entrada: Un esquema de red N y un query Q referido a un conjunto deatributos X.

```
procedure PQ(N, Q, X)

begin
    S <- Ø
    for cada registro tipo Ri de N do
        if X⊆R̄i
          then
            S <- S U {Ri}
        endif
    endfor
    for cada Ri de S do
        if existe Rj ∈ S tal que Rj sea ancestro de Ri
          then
            S <- S - {Ri}
        endif
    endfor
    for cada Ri de S do
        call OPT-A(N, Ri, X, Opath) [1]
        call GEN-COD(Q, Opath)
    endfor
end
```

Comentarios del algoritmo

[1]: O bien, call OPT-B(N, R_i, X, Opath)