CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

"OPTIMACION DE CONSULTAS A BASE DE DATOS BASADA EN REGLAS"

Tesis que presenta el Ing. César Alejandro Galindo Legaria para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERIA ELECTRICA con opción de COMPUTACION. Trabajo dirigido por el Dr. Renato Barrera Rivera.

Becario de CONACYT. México D. F., 25 de Septiembre de 1987.

Introducción

Un componente importante de los sistemas de base de datos es el optimador de consultas. Esto se debe a que pueden usarse lenguajes de consulta de muy alto nivel, que proporcionan operaciones abstractas muy fáciles de usar y entender pero cuya ejecución puede ser costosa. El papel del optimador es generar un plan de acceso eficiente para obtener ciertos datos, a partir de una consulta formulada por un usario.

Esta importancia del optimador se manifiesta a partir de la propuesta del modelo relacional y su lenguaje de consulta [15]. A partir de entonces se han desarrollado lenguajes de consulta de muy alto nivel para sistemas de base de datos relacionales, así como métodos de optimación para encontrar planes de acceso eficientes a partir de una consulta dada. El problema es complejo, y el término "optimador" no es exacto; pero ese es el nombre que se da comunmente a estos programas, que no necesariamente ecuentran el mejor plan de ejecución para cada consulta, sino solamente un plan que sea razonablemente "bueno".

Recientemente, se han venido desarrollando nuevas propuestas sobre modelos de base de datos, con otro tipo de operandos y operaciones [6,17,32] y en las que el optimador juega un papel importante para una implantación eficiente. Se habla incluso de bases de datos en las que se puedan agregar nuevos tipos de objetos con sus operaciones propias. En un sistema de este tipo, como ya se ha señalado [11,19,23,29], se requiere de un optimador extensible: deben poder agregarse nuevos objetos y operaciones; nuevos métodos para llevarlas a cabo, con estructuras y costos propios; y sugerencias en cuanto a cómo efectuar la optimación.

En este trabajo se propone la arquitectura de un optimador general extensible basado en reglas (OG). Dada la experiencia que existe en la optimación de consultas para el modelo relacional, se ilustra la arquitectura general del OG construyendo un optimador para bases de datos relacionales distribuidas (ORD).

En el capítulo 1 se presentan los conceptos básicos del modelo relacional, junto con las técnicas de optimación propias del mismo; en el capítulo 2, conceptos básicos y técnicas de optimación para una base de datos distribuida que usa el modelo relacional. En el capítulo 3 se presentan los principios generales de la optimación basada en reglas, así como la arquitectura del OG. En el capítulo 4 se describen en detalle los módulos que componen al ORD. Finalmente, en el capítulo 5 se hace una comparación del OG y ORD con otros trabajos. Se asume el conocimiento de cierta terminología de gráficas que puede consultarse en [2].

1. EL MODELO DE BASE DE DATOS RELACIONAL

La propuesta del modelo relacional [15] trae una serie de cambios importantes en la manera de ver y usar las bases de datos. Entre otras cosas, se propone un modelo de datos uniforme y teóricamente manejable, y se hace énfasis en la independencia entre el lenguaje de consulta y las estructuras de almacenamiento de los datos. A partir de entonces se han desarrollado sistemas de base de datos que siguen este modelo, se ha adquirido experiencia en su uso y luego, a partir de esta experiencia, se han propuesto otros modelos. Por esto el modelo relacional es un buen punto de partida para el estudio de las bases de datos, y también para realizar las primeras pruebas en cuanto a optimación basada en reglas.

En este capítulo se presentan los conceptos básicos de un sistema de base de datos, y se repasa brevemente el modelo relacional. Para más información se puede consultar [34].

1.1. Conceptos básicos

Para facilitar el uso de una base de datos se manejan varios niveles de abstracción. Cada nivel corresponde a una forma de ver la misma información, de acuerdo con un propósito. El nivel físico especifica cómo deben verse los datos en cuanto a su almacenamiento; esto es, qué estructuras deben usarse para guardarlos de manera eficiente y confiable. El nivel conceptual especifica cómo se pueden presentar los datos al usuario para que los entienda fácilmente, y se debe usar algún formato que ponga de relieve las relaciones conceptuales entre ellos. Estos niveles se ilustran a continuación.

> Ejemplo 1.1. Supóngase que en una base de datos guardamos, entre otras cosas, información acerca de los libros que se tienen en una biblioteca. En el nivel físico, se puede ver que los datos de los libros están almacenados en un archivo que tiene un índice sobre el campo autor y otro sobre el campo numlibro. En el nivel conceptual, por otro lado, los datos se visualizan en la forma de una tabla en cuyos renglones aparece la información de cada libro. Esto se esquematiza en la Fig. 1.1.

El uso de estos niveles de abstracción plantea un problema importante: El usuario describe sus consultas según la manera como él ve los datos (nivel conceptual), pero éstas deben ser resultas en términos del nivel físico. Para cada consulta del usuario debe determinarse un plan de acceso que obtenga los datos solicitados, de acuerdo con la manera en que están almacenados.

Al componente que se encarga de encontrar un plan de acceso eficiente para resolver una consulta dada se le llama optimador. Este módulo realiza su tarea guiado por heurísticas que se han desarrollado a lo largo de los años. Mientras mayor sea la diferencia entre los niveles físico y conceptual y mientras más abstractos sean los lenguajes de consulta que se usen, más

(a) Nivel físico: archivos de datos.

titulo	autor	nombree	numlibro
Data Structures and Algorithms	Aho	Addison-Wesley	8476
Principles of Compiler Design	Aho	Addison-Wesley	3645
Distributed Databases	Ceri	McGraw-Hill	9087
Programming in Prolog	Clocksin	Springer-Verlag	1324
Artificial Intelligence	Rich	McGraw-Hill	5789
Artificial Intelligence	Winston	Addison-Wesley	3867

(b) Nivel conceptual: tablas de datos.

Fig. 1.1. Niveles de abstracción.

importante es la tarea del optimador, que nos debe llevar de la facilidad de expresión a la eficiencia de ejecución.

Hay varios enfoques en cuanto a la manera en que deben verse los datos en el nivel conceptual; estos enfoques reciben el nobre de modelos de datos. Un modelo de datos se compone de dos partes: una notación matemática para describir los datos y sus relaciones, y un conjunto de operaciones para manipularlos. Aunque el objetivo principal es la claridad y facilidad de uso, los modelos de datos propuestos están influenciados por alguna estructura de almacenamiento físico.

1.2. El modelo relacional

Después de la propuesta inicial del modelo relacional, se han presentado varias versiones de sus operadores que, aunque son muy semejanntes, guardan ciertas diferencias en notación o significado. En lo que sigue nos basaremos en la notación usada en [13].

1.2.1. La relación

En el modelo relacional los datos aparecen al usuario en la forma de tablas. Cada tabla tiene un formato fijo en cuanto al numero de columnas y tipo de datos que almacena. Típicamente, una base de datos está compuesta por varias tablas. Cada tabla tiene un número variable de renglones, que no se duplican, y que

libros:

titulo	autor	nombree	numlibro
Data Structures and Algorithms	Aho	Addison-Wesley	8476
Principles of Compiler Design	Aho	Addison-Wesley	3645
Distributed Databases	Ceri	McGraw-Hill	9087
Programming in Prolog	Clocksin	Springer-Verlag	1324
Artificial Intelligence	Rich	McGraw-Hill	5789
Artificial Intelligence	Winston	Addison-Wesley	3867

editoriales:

nombree	direcce	ciudade
Addison-Wesley	1027 5th Av	Nueva York
McGraw-Hill	558 7th Av	Nueva York
Springer-Verlag	879 3rd Av	Nueva York

usuarios:

nombre	direccion	ciudad	numusr
García	Fresnos #528	Querétaro	196
López	J.S.Bach # 55	Mexico D.F.	435

prestamos:

numusr	numlibro	fecha
196	1324	1/8/87
435	8476	3/8/87
196	5789	7/8/87

Fig. 1.2. Una base de datos en tablas.

pueden agregarse o eliminarse. La claridad de este tipo de representación se aprecia en el siguiente ejemplo.

Ejemplo 1.2. Queremos guardar en una base de datos la información concerniente a una biblioteca. Nos interesa guardar información sobre los libros que tenemos y sus editoriales, nuestros usuarios y qué libros les hemos prestado. Determinamos qué datos nos interesa registrar de cada libro, usuario, editorial y préstamo que se realice, y de acuerdo con el modelo relacional ordenamos los datos en tablas. Nuestras tablas resultantes aparecen en la Fig. 1.2.

La estructura de las tablas tiene semejanza con un concepto matemático importante, que constituye la base teórica del modelo: la relación. Una relación entre varios conjuntos o dominios D1..Dk es un conjunto de eneadas de k componentes en las que el

componente i ha sido tomado del dominio Di. En términos de nuestro ejemplo anterior, estamos representando en la tabla libros una relación sobre los conjuntos autores, titulos, nombres de editorial y numeros de libro.

El orden en que aparecen los componentes en una eneada es significativo, y corresponde al orden en que consideramos a los dominios en la relación. Podemos, sin embargo, remplazar cada eneada por un mapeo de nombres a valores (nombre de componente y valor de componente), y hacer irrelevante la posición de un componente, siempre que éste puede identificarse. Así pues, una relación es un conjunto de elementos donde cada elemento es un conjunto de valores con un nombre asociado. Esto corresponde a la idea de una tabla compuesta de renglones que no se repiten y que tienen valores para columnas etiquetadas, y también a la idea de un archivo compuesto de registros que no se repiten y en el que el orden de declaración de sus campos no es importante.

Por la manera como se construye una relación, llamamos eneada a cada renglón de su tabla, y decimos que su cardinalidad es el número de eneadas que contiene. Llamamos atributos o campos de una relación a los valores que aparecen en cada columna de la tabla, así como a las propias etiquetas de columna.

Para construir la base de datos, hay que especificar primero sobre qué conjuntos van a estar construidas las relaciones y cómo se van a llamar, para después manejar su contenido agregando o eliminando renglones de las tablas. A esta información se le llama el esquema de la base de datos, y corresponde a formatos de tablas vacías. Las relaciones mismas, el contenido de las tablas en un momento dado es lo que se llama una instancia de la base de datos. En la Fig. 1.3. se muestran los dominios, relaciones y esquemas relacionales del ejemplo 1.2.; lo que aparece en la Fig. 1.2. es una instancia de nuestra base de datos.

A una base de datos se pueden asociar predicados que nos interesa que se satisfagan para cualquier instancia de la misma. Dichos predicados reciben el nombre de restricciones de integridad porque restringen las instancias que puede tener la base de datos a que tengan cierto "sentido", en términos de nuestra interpretación. Una restricción de integridad para nuestro ejemplo de la biblioteca puede ser la siguiente: las fechas registradas en la relación préstamos no deben ser mayores a la fecha del día de hoy.

Dominios:

titulo: Título de un libro.
autor: Autor de un libro.
numlibro: Número de clasificación de un libro.
nombree: Nombre de una editorial.
direcce: Dirección de una editorial.
ciudade: Ciudad en donde está una editorial.
nombre: Nombre de un usuario de la biblioteca.
direc: Dirección de un usario de la biblioteca.
ciudad: Ciudad donde vive un usuario de la biblioteca.
numusr: Numero de registro de un usuario.
fecha: Fecha en la que se realiza un préstamo.

Relaciones:

libros: Información sobre los libros que hay en la biblioteca.

$$libros \subseteq titulo \times autor \times nombree \times numlibro$$

editoriales: Información sobre editoriales de libros de la biblioteca.

$$editoriales \subseteq nombree \times direcce \times ciudade$$

usuarios: Información sobre los usuarios de la biblioteca.

$$usuarios \subseteq nombre \times direc \times numusr$$

prestamos: Información sobre los libros prestados a usuarios.

$$prestamos \subseteq numusr \times numlibro \times fecha$$

Esquemas relacionales:

libros(titulo,autor,nombree,numlibro)

editoriales(nombree,direcce,ciudade)

usuarios(nombre,direc,ciudad,numusr)

prestamos(numusr,numlibro,fecha)

Fig. 1.3. Dominios, relaciones y esquemas relacionales de una base de datos.

1.2.2. Operadores relacionales

Con las relaciones recién definidas se pueden realizar ciertas operaciones cuyo resultado son nuevas relaciones. Estas operaciones corresponden a operadores relacionales con los que se construyen expresiones relacionales.

En seguida se describen los cinco operadores relacionales básicos. Las variables R, S y T representan relaciones, la variable C representa un conjunto de nombres de campos, y la variable F representa una fórmula que involucra nombres de campos, constantes, comparadores (=, < , >) y el conectivo lógico de conjunción and. Las fórmulas se restringen a este esquema por simplicidad; también podrían considerarse los conectivos or y not, además de otros comparadores. La Fig. 1.4. (a-c) muestra valores ejemplo para las relaciones R, S y T.

La selección SL(F):R produce una relación con los mismos campos que R y cuyas eneadas son las eneadas de R que satisfacen la fórmula F. En la Fig. 1.4. (d) se muestra un ejemplo de selección.

La proyección PJ(C):T produce una relación que tiene los campos especificados por C (debe ser un subconjunto de los campos de T) y cuyas eneadas se construyen a partir de las de T, eliminando los campos que no aparecen en C. La cardinalidad del resultado puede ser menor que la cardinalidad de T, porque las eneadas repetidas sólo aparecen una vez. En la Fig. 1.4. (e) se muestra un ejemplo de proyección.

La unión UN:R,T sólo tiene sentido cuando las relaciones R y T tienen los mismos campos. La unión produce una relación con los mismos campos que los de las relaciones que constituyen su argumento, y contiene todas las eneadas que aparecen en R o en S. En la Fig. 1.4. (f) aparece un ejemplo de unión.

La diferencia DF:R,T sólo tiene sentido cuando las relaciones R y T tienen los mismos campos. La diferencia produce una relación con los mismos campos que los de las relaciones que constituyen su argumento, y contiene las eneadas que aparecen en R pero no en S. En la Fig. 1.4. (g) aparece un ejemplo de diferencia.

El producto cartesiano PD:R,S opera con relaciones en las que los nombres de campos de una no aparecen en la otra. Produce una relación que tiene los campos de ambas relaciones, y sus eneadas se forman combinando cada eneada de R con todas las de S. La cardinalidad del resultado es el producto de las cardinalidades de las relaciones sobre las que se aplica. En la Fig. 1.4. (h) se muestra un ejemplo de producto.

Consideramos ahora algunos operadores compuestos que se definen en términos de los anteriores; corresponden a secuencias de operaciones frecuentemente usadas para las que existen métodos de ejecución eficientes.

La junta JN(F):R,S se formula en terminos de los operadores anteriores como SL(F):PD:R,S. En la Fig. 1.4. (i) aparece un ejemplo de junta.

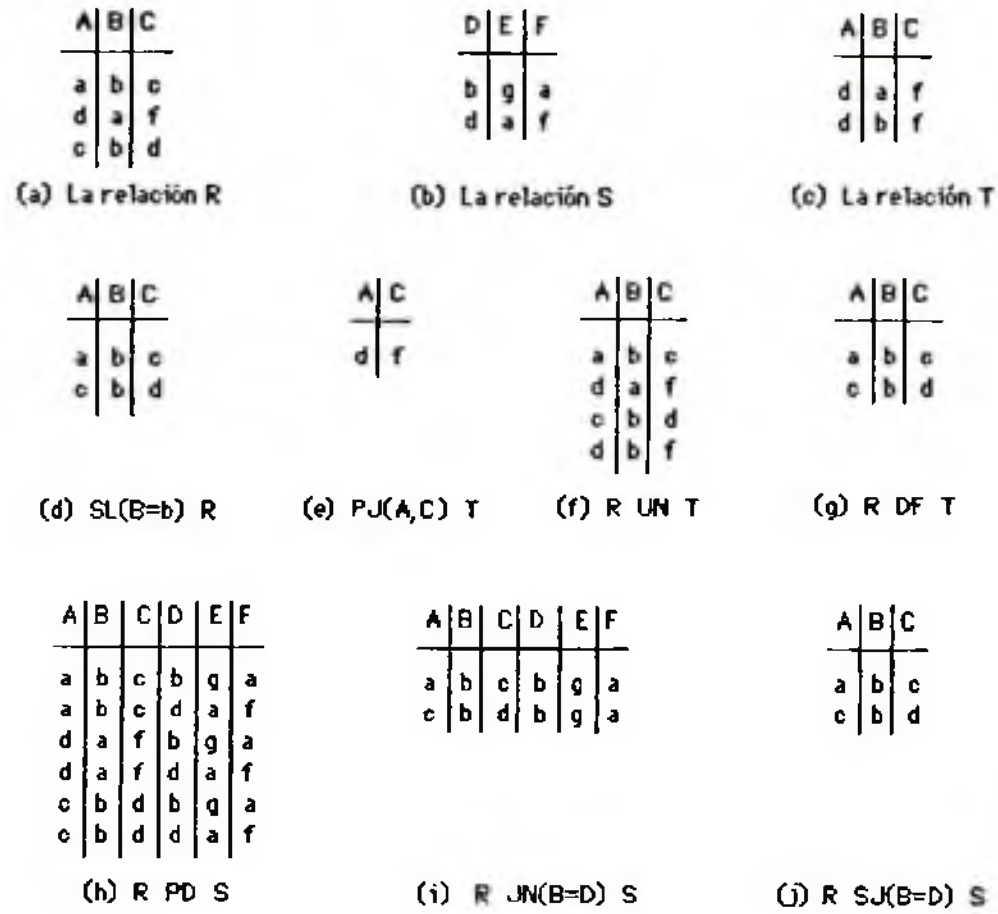


Fig. 1.4. Operadores del álgebra relacional.

La semijunta SJ(F):R,S corresponde a PJ(atr(R)):JN(F):R,S, donde atr(R) son los campos de R. Esta operación tiene el propósito de reducir la cardinalidad de la relación R antes de que sea operada con S mediante una junta. Como se verá en el capítulo siguiente, se usa para resolver consultas en base de datos distribuidas. En la Fig. 1.4 (j) se muestra un ejemplo de semijunta.

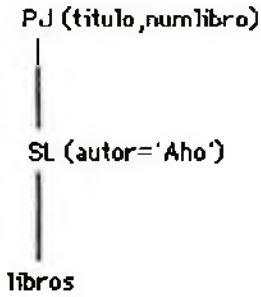
Para una lista de las propiedades algebraicas a las que obedecen los operadores relacionales, así como los métodos de demostración de éstas, se puede consultar [12,34].

1.2.3. Lenguajes de consulta

Los sistemas relacionales de base de datos proporcionan lenguajes de consulta; estos lenguajes deben tener, al menos, la misma capacidad expresiva del álgebra relacional para considerarse completos [16,34]. Un lenguaje de consulta cuyo uso se ha extendido mucho es el SQL [5], que tiene características que lo hacen "más que completo": algunas consultas en SQL pueden

```
select titulo,numlibro
from   libros
where  autor='Aho'
```

(a) Consulta 1 en SQL.



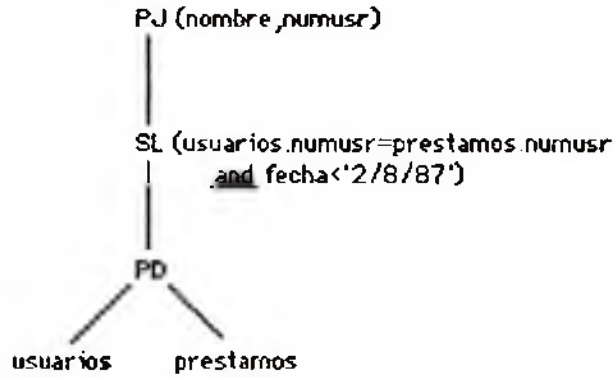
(b) Consulta 1 en álgebra relacional

titulo	numlibro
Data Structures and Algorithms	8476
Principles of Compiler Design	3645

(c) Resultado de la consulta 1.

```
select nombre,numusr
from   usuarios,prestamos
where  usuarios.numusr=prestamos.numusr
  and  fecha<'2/8/87'
```

(d) Consulta 2 en SQL.



(e) Consulta 2 en álgebra relacional

nombre	numusr
García	196

(f) Resultado de la consulta 2.

Fig. 1.5. Consultas a una base de datos.

no tener equivalente en álgebra relacional. Con un uso restringido de SQL, sin embargo, sí se pueden construir expresiones equivalentes en álgebra. A continuación se muestran algunas consultas sobre la base de datos del ejemplo 1.2.

Ejemplo 1.3. Consulta 1. Para nuestra base de datos de la biblioteca, queremos saber qué libros tenemos del autor 'Aho'. Podemos formular la consulta en SQL como se muestra en la Fig. 1.5. (a); esta consulta se puede traducir luego a la expresión en álgebra relacional que aprece en (b); en (c) aparece la respuesta de la consulta, de acuerdo con la instancia de la base de datos de la Fig. 1.2.
Consulta 2. Queremos saber qué usuarios tienen libros que hayan tomado prestados antes del 2/8/87. En la Fig. 1.5. (d) se muestra la consulta en SQL, en (e) la expresión algebraica equivalente, y en (f) el resultado de la consulta.

1.3. Técnicas de optimación

En la sección anterior vimos cómo las consultas del usuario pueden expresarse mediante álgebra relacional, que define un procedimiento para obtener los datos solicitados. Hay ciertas transformaciones que aplicadas sobre una expresión relacional producen otra equivalente (que da el mismo resultado que la primera bajo cualquier instancia de la base de datos), pero cuyo tiempo de ejecución puede ser menor. En la práctica, resulta que las transformaciones pueden conducir a mejoras de varios órdenes de magnitud en el tiempo de ejecución. En esta sección vemos algunas de estas transformaciones. Suponemos que cada relación está almacenada en un archivo, y que puede tener índices sobre algunos de sus campos.

En [26] se puede encontrar un resumen muy completo de técnicas de optimación, aunque el enfoque de proceso de consultas seguido en la referencia es distinto del usado aquí.

1.3.1. Expresiones algebraicas extendidas

Aunque una expresión algebraica define un procedimiento para obtener un resultado, las operaciones del álgebra relacional se definen sobre conjuntos abstractos y no sobre estructuras de almacenamiento. Dependiendo de la manera en que estén almacenadas las relaciones en el sistema, cada operador relacional puede ejecutarse mediante varios métodos. Formamos una expresión algebraica extendida asociando a cada operador relacional que aparece en una expresión algún método de ejecución. La expresión extendida es útil porque define un plan de acceso y por lo tanto puede evaluarse su costo de ejecución, además de conservar la estructura algebraica de la consulta. Llamaremos simplemente expresiones a estas expresiones algebraicas extendidas, cuando no se preste a confusión.

Una primera transformación que aparece naturalmente en estas expresiones es la de cambio de método: para una operación relacional dada, podemos usar otro método de ejecución sin alterar el resultado de la expresión, pero sí quizá modificando el costo de evaluación. Este tipo de transformación se ilustra en el siguiente ejemplo.

> Ejemplo 1.4. Para éste y los ejemplos siguientes que usan la base de datos de la biblioteca, supondremos que el contenido de la misma está caracterizado por los parámetros que se muestran en la Fig. 1.6.
> En la Fig. 1.7. (a), se muestra la expresión extendida inicial para la Consulta 1 del ejemplo 1.3. El plan de acceso indicado es recorrer secuencialmente el archivo de la relación libros y seleccionar las eneadas cuyo campo autor sea igual a 'Aho'; de acuerdo con el perfil mostrado en la Fig. 1.6., este recorrido implicará leer 99610 páginas de memoria secundaria.
> Una transformación del tipo cambio de método, aplicada sobre la expresión de la Fig. 1.7. (a), nos lleva a la expresión de la Fig. 1.7. (b). Como existe un índice sobre el campo

Relación libros

Cardinalidad: 1000000
Indices: {numlibro, autor}

Campo	Ancho	Valores distintos
titulo	15	900000
autor	15	800000
nombree	15	500
numlibro	6	1000000

Relación prestamos

Cardinalidad: 40000
Indices: {numlibro, numusr}

Campo	Ancho	Valores distintos
numlibro	6	20000
numusr	6	40000
fecha	6	300

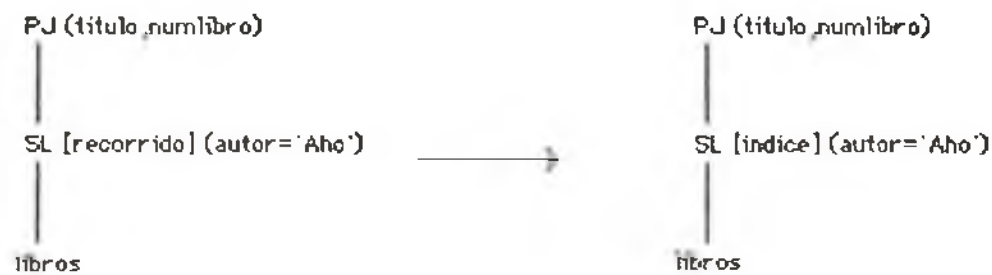
Relación usuarios

Cardinalidad: 40000
Indices: {numusr}

Campo	Ancho	Valores distintos
nombre	15	40000
direo	15	30000
ciudad	10	50
numusr	6	40000

Octetos por página: 512

Fig. 1.6. Perfiles de las relaciones de una base de datos.



(a) Consulta inicial. (b) Consulta transformada.

Fig. 1.7. Una transformación por cambio de método.

autor, otro método para resolver la consulta es accesar directamente las eneadas haciendo uso de él; de acuerdo con el perfil de la relación, se estima que esto involucrará la lectura de una sola página del archivo principal. El plan de ejecución aparece en el apéndice 1.

1.3.2. Reducción de resultados intermedios

Una de las primeras heurísticas de optimación de consultas es la reducción del tamaño de los resultados intermedios [33]. Si se requiere el almacenamiento temporal de un resultado intermedio, son necesarios menos accesos a memoria secundaria si el tamaño de la relación a almacenar es menor. Esta reducción corresponde a la aplicación de operadores unarios que reducen el tamaño de una relación "tan pronto como sea posible". En términos de transformaciones, se trata de usar la propiedad distributiva de los operadores unarios respecto de los binarios para que los unarios se realicen primero, tantas veces como sea posible.

Las transformaciones de este tipo conducen a reducciones importantes en el tiempo de ejecución, independientemente del método que se use para implantar cada operación, y dependen completamente de las propiedades algebraicas de los operadores relacionales, así que las llamaremos transformaciones algebraicas. Más aun, como conducen siempre a un tiempo de ejecución menor, las consideraremos como "reglas de mejora". En el siguiente ejemplo se ilustra este tipo de transformación.

Ejemplo 1.5. Tomemos la Consulta 2 del ejemplo 1.3. Nos pide realizar un producto cartesiano entre relaciones, lo que produce una relación más grande que sus argumentos, luego una selección y finalmente una proyección. La expresión correspondiente aparece en la Fig. 1.8. (a). De acuerdo con el perfil de la Fig. 1.6., la relación usuarios ocupa 3594 páginas, mientra que préstamos ocupa 1407; y el producto cartesiano tomará 1407+1407*3594 = 5858165 accesos a memoria secundaria.
Examinando las condiciones de la selección, descubrimos que una de ellas se refiere solamente a la relación préstamos (fecha<'2/8/87'). Con base en la heurística de reducción de resultados intermedios, aplicamos esta selección antes de que se realice el producto. Después reconocemos el patrón selección-producto y lo remplazamos por una operación de junta, que tiene métodos eficientes de ejecución. La expresión resultante se muestra en la Fig. 1.8. (b). Suponiendo que la selección reduce a la mitad la cardinalidad de préstamos, y que tanto la selección como la junta se ralizan de la forma más simple (no hacemos cambios de método), el cálculo de la junta tomará 1407+704*3594 = 2531583 accesos a memoria secundaria; menos de la mitad de las que se requieren con el procedimiento anterior.
Si elegimos un método de junta que tome en cuenta el índice sobre numusr en usuarios, el número de páginas que se necesita accesar para resolver la consulta se reduce a sólo

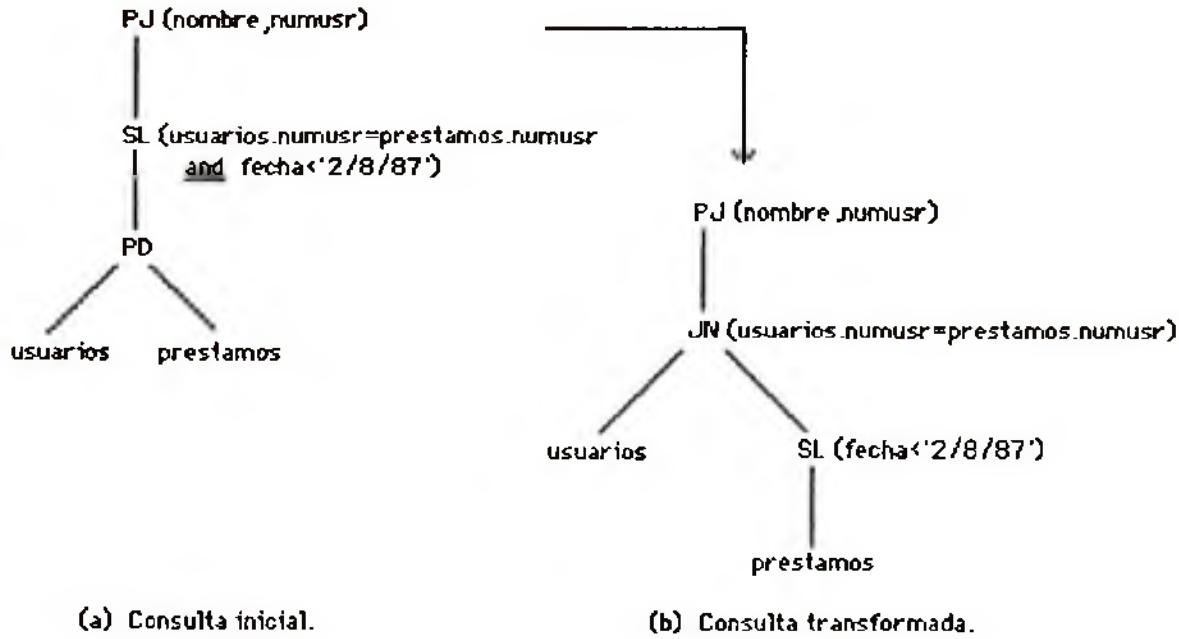


Fig. 1.8. Una transformación algebraica.

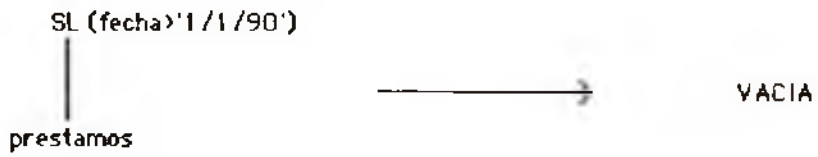
19847. El plan de ejecución correspondiente aparece en el apéndice 1.

La aplicación de la heurística anterior está sujeta a una restricción: no se puede saber si en verdad conviene la pronta aplicación de un operador unario si éste se aplica sobre una relación con índices. La razón de esta restricción es que los índices de un archivo pueden usarse para resolver eficientemente operaciones de junta. En este caso debemos recurrir a nuestro modelo de costos para probar si es conveniente o no la transformación.

1.3.3. Simplificación basada en información semántica

Otro tipo de transformaciones que pueden realizarse sobre una expresión se apoyan en las inferencias que podamos hacer con base en las restricciones de integridad de la base de datos [28]. Estas inferencias pueden conducir a cambios estructurales radicales en una expresión de consulta, pero en su forma más general requieren de herramientas de análisis sofisticadas.

En este trabajo se incluye un módulo sencillo para el análisis de predicados restringidos. Este módulo de análisis realiza las tareas de normalización y detección de predicados contradictorios. La detección de predicados contradictorios permite descubrir resultados intermedios vacíos, que luego conducen a importantes simplificaciones en la consulta. La normalización permite la detección de cláusulas redundantes, que pueden llevar a la eliminación de operaciones innecesarias. Un ejemplo sencillo del uso de información semántica se presenta en la figura siguiente.



(a) Consulta inicial. (b) Consulta transformada.

Fig. 1.9. Una transformación que usa información semántica.

Ejemplo 1.6. Suponga que en una consulta necesitamos evaluar la selección de los préstamos realizados después del 1/1/90, que se muestra en la Fig. 1.9. (a). Haciendo uso de la restricción de integridad pertinente (no puede haber préstamos registrados con una fecha posterior al día de hoy), detectamos una contradicción y descubrimos que el resultado es una relación vacía, Fig. 1.9. (b).

2. EL MODELO RELACIONAL EN BASE DE DATOS DISTRIBUIDAS

En ocasiones es conveniente no tener almacenada toda una base de datos en una sola máquina. Cuando la información se almacena por partes en varias máquinas, se dice que tenemos una base de datos distribuida. En bases de datos distribuidas, algunas consultas necesitan de la transmisión de datos (relaciones o porciones de éstas) entre las distintas máquinas que almacenan la información, y que deben estar conectadas mediante una red. Aunque la velocidad de transmisión de datos dependa del tipo de red (local o geográfica), el tiempo que se lleva la transmisión es usualmente mayor que el tiempo de acceso a memoria secundaria, por lo que se le considera el costo dominante.

En este capítulo se presentan conceptos generales de base de datos distribuidas, y luego se muestran las técnicas de optimación usadas para resolver consultas en este contexto. Para más información sobre el tema se puede consultar [13].

2.1. Conceptos básicos

Una base de datos distribuida se caracteriza porque los datos que la componen no están almacenados en una sola máquina sino en varias, colocadas en distintos sitios y comunicadas mediante una red. Esta configuración se muestra en la Fig. 2.1. En principio se puede usar cualquier modelo de datos en base de datos distribuidas; el más estudiado, sin embargo, es el modelo relacional, que también nosotros asumiremos.

Una relación no necesita estar almacenada en su totalidad en un solo sitio, la podemos partir en fragmentos que se almacenan independientemente. Cada uno de estos fragmentos puede incluso estar duplicado en varios sitios. Hay varios tipos de fragmentación; en este trabajo sólo consideraremos la fragmentación horizontal, en la que cada eneada de una relación global va a parar, completa, a uno de los fragmentos. Todas las eneadas de una relación global se almacenan en algún fragmento. Vamos a suponer, adicionalmente, que una eneada no se almacena en más de un fragmento (a menos que se trate de una duplicación del fragmento completo). Una relación sigue cierto criterio de fragmentación para dividirse, que se diseña con el propósito de que la mayor parte de las operaciones se realicen localmente. Para cada relación a fragmentar determinamos, entonces, un conjunto de predicados que van a corresponder a cada fragmento. La disyunción de todos los predicados coincide con las restricciones de integridad de la relación, y la conjunción de dos predicados distintos cualesquiera debe ser una contradicción. Cada predicado del conjunto se convierte en la restricción de integridad del fragmento correspondiente. A la relación que se fragmenta podemos llamarla relación global, y a cada uno de sus fragmentos relación local.

Independientemente de la distribución que se ha hecho de una relación global, nos interesa seguir realizando consultas sobre ella, de modo que el sistema de base de datos se encargue de

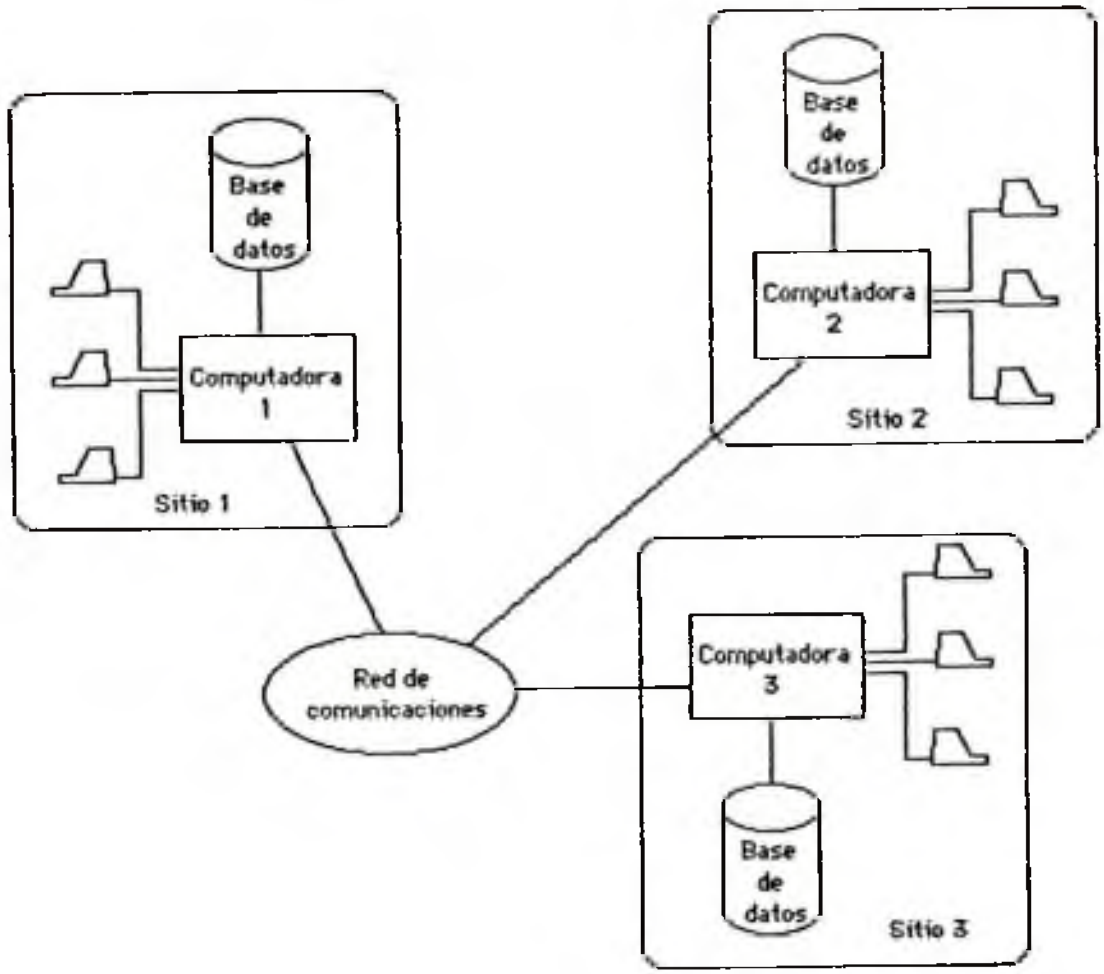


Fig. 2.1. Un sistema para base de datos distribuidas.

realizar las transmisión de los datos que sean necesarios para resolverlas. La transformación de una consulta global a otra que haga referencia sólo a fragmentos, se realiza remplazando cada referencia a una relación global por la expresión relacional que la calcula, en base a sus fragmentos. En el ejemplo siguiente se ilustran estos conceptos.

> Ejemplo 2.1. Con referencia al ejemplo 1.2. de una base de datos para una biblioteca, supongamos ahora que tenemos un "sistema de bibliotecas". Hay cinco bibliotecas que se identifican, cada una, por un número. Las bibliotecas 1, 2 y 3 se encuentran todas en una zona relativamente pequeña, y la más grande de ellas es la 1; decidimos almacenar en ésta la información de los libros que tenemos en ella y las otras dos. Las bibliotecas 4 y 5 se encuentran más lejos de esta zona y decidimos almacenar en cada una de ellas la información sobre los libros que poseen. Esto nos lleva a tener tres relaciones locales libros1, libros2 y libros3 que son fragmentos de una relación global libros. En la Fig. 2.2. (a) se muestran las relaciones locales basadas en los datos del ejemplo 1.2., junto con su criterio de fragmentación.
> Para calcular la relación global libros, tenemos que realizar la unión de los fragmentos que la componen, como se muestra en la Fig. 2.2. (b).

Aunque los métodos para realizar las operaciones relacionales son los mismos que en una base de datos centralizada, cada operaciones debe ejecutarse en una máquina dada. En nuestro ejemplo anterior, para calcular la unión tenemos que transmitir las relaciones a un sitio común antes de que ésta se realice.

2.2. Técnicas de optimación

El costo de transmisión en la ejecución de consultas, en este tipo de base de datos, es el factor más importante. En bases de datos distribuidas se aplican todas las técnicas de optimación usadas en base de datos centralizadas, además de algunas otras que tienen el propósito de reducir el volumen de dados que se necesita transmitir para resolver una consulta.

En esta sección se presentan las técnicas de optimación propias de base de datos distribuidas, y cuyo propósito es reducir los costos de transmisión. En [37] se puede encontrar un resumen más completo de técnicas de optimación para base de datos distribuidas.

2.2.1. Uso de cualidades de las relaciones

A las restricciones de integridad que se aplican a una relación (en especial a un fragmento) se les llama tambien cualidades de la relación. La simplificación de expresiones basada en información semántica (véase 1.3.3.) se sugiere en el

libros1(titulo,autor,nombree,numlibro,numbiblioteca)

titulo	autor	nombree	numlibro	numbiblioteca
Data Structures and Algorithms	Aho	Addison-Wesley	8476	1
Principles of Compiler Design	Aho	Addison-Wesley	3645	2

Criterio de fragmentación: numbiblioteca<4

libros2(titulo,autor,nombree,numlibro,numbiblioteca)

titulo	autor	nombree	numlibro	numbiblioteca
Distributed Databases	Ceri	McGraw-Hill	9087	4

Criterio de fragmentación: numbiblioteca=4

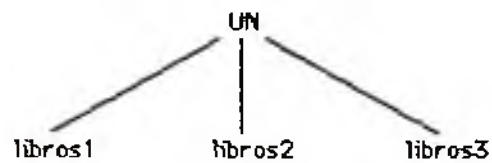
libros3(titulo,autor,nombree,numlibro,numbiblioteca)

titulo	autor	nombree	numlibro	numbiblioteca
Programming in Prolog	Clocksin	Springer-Verlag	1324	5
Artificial Intelligence	Rich	McGraw-Hill	5789	5
Artificial Intelligence	Winston	Addison-Wesley	3867	5

Criterio de framentación: numbiblioteca=5

(a) Fragmentos de la relación.

libros(titulo,autor,nombree,numlibro,numbiblioteca)



(b) Recuperación de la relación fragmentada.

Fig. 2.2. Una relación fragmentada.

contexto de bases de datos distribuidas para eliminar transmisiones de datos innecesarias. El principio es el mismo, pero se incluye como una técnica de base de datos distribuida porque ha sido subrayada como tal [12,13]. A continuación se ilustra el papel de este tipo de simplificación en el proceso de una consulta distribuida.

Ejemplo 2.2. Con referencia a la Consulta 1 del ejemplo 1.3., queremos saber qué libros del autor 'Aho' tenemos en la biblioteca 4, con base en nuestra relación global libros del ejemplo 2.1. La expresión correspondiente a nuestra consulta se encuentra en la Fig. 2.3. (a). El primer paso para procesar la consulta es remplazar la relación global por una expresión que la calcula, Fig. 2.3. (b). De aquí, usaremos el criterio de reducción de resultados intermedios (véase 1.3.2.) para llegar a la expresión de la Fig. 2.3. (c). Realizando un análisis de predicados que considera las cualidades de las relaciones y las operaciones de selección que estamos realizando, descubrimos que algunos de los resultados intermedios son vacíos, y llegamos a la expresión simplificada de la Fig. 2.3. (d). Finalmente, y otra vez gracias al análisis de predicados, descubrimos una condición redundante en la selección, y al eliminarla llegamos a la expresión de la Fig. 2.3. (e).

2.2.2. Uso de la operación semijunta

La técnica de optimación más importante para la ejecución de consultas distribuidas lo consituye el uso de la operación semijunta [8]. Esta operación proporciona una alternativa a la ejecución de una operación de junta entre relaciones que se encuentran en distintos sitios.

Una primera estrategia para realizar una junta entre las relaciones R y S bajo una condición F, si las relaciones se encuentran en sitios distintos, es transmitir la relación R al sitio de S y allí llevar a cabo la semijunta (o viceversa). Pero quizá transmitiendo sólo un subconjunto de R podamos calcular la junta que se pide; la operación semijunta (véase 1.2.2.) nos dice exactamente cuál es el subconjunto mínimo de R que debemos usar para calcular la junta. Para calcular la semijunta en cuestión necesitamos transmitir la proyección de los atributos de S que toman parte en la condición F al sitio de R, así que la solución de una junta por medio de una semijunta involucra dos transmisiones: una transmisión de S a R que tiene el propósito de reducir a la relación R, y otra de R a S para transmitir la relación ya reducida y ejecutar la operación de junta. El esquema de ejecución es simétrico, y el costo de cada alternativa es distinto. Esta estrategia de ejecución de juntas no necesariamente reduce los costos de transmisión, pero puede llegar a reducirlos considerablemente. La estrategia se ilustra en la Fig. 2.4.

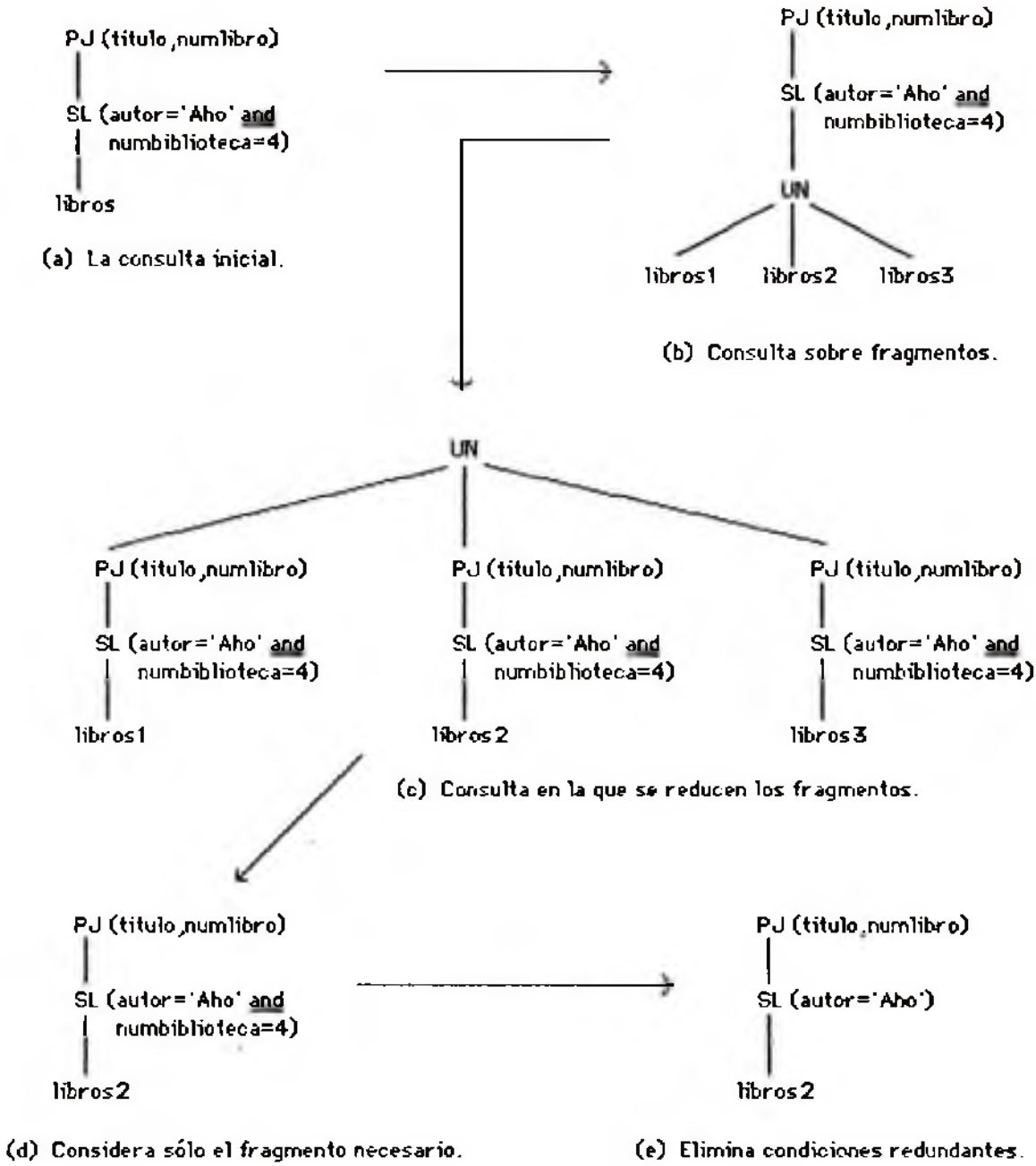
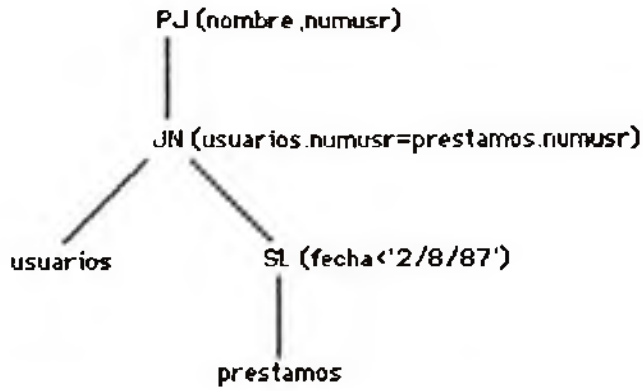
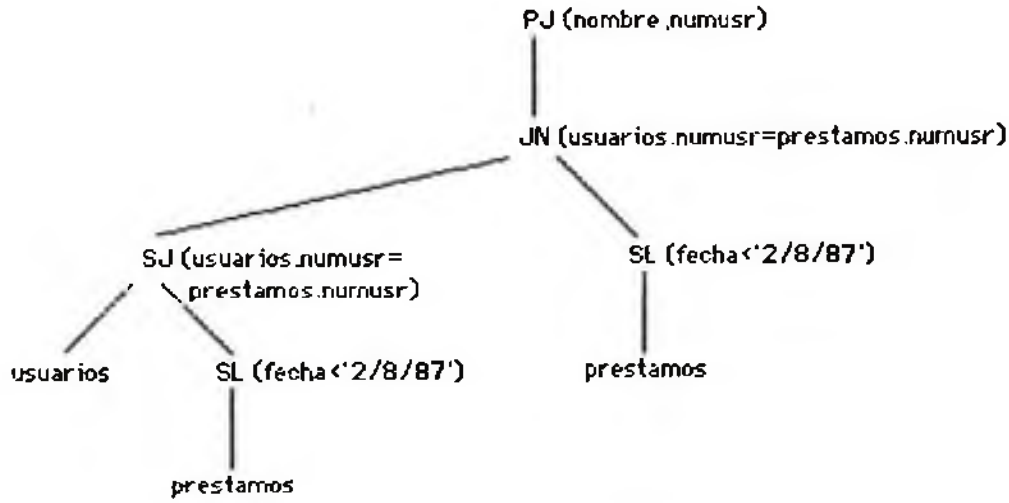


Fig. 2.3. Proceso de una expresión con relaciones fragmentadas.



(a) Una expresión que involucra la junta de relaciones.



(b) Evaluación de la junta usando una semijunta.

Fig. 2.4. Uso de la operación semijunta.

Ejemplo 2.3. Con referencia a la Consulta 2 del ejemplo 1.3., consideremos que las relaciones usuarios y préstamos están almacenadas en sitios distintos, digamos 1 y 2, respectivamente, y que la respuesta se requiere en el sitio 2. La expresión de la consulta se encuentra en la Fig. 2.4. (a). Su ejecución es como sigue:

1. Calcula la selección que se pide sobre la relación préstamos, el resultado permanece en el sitio 2.
2. Transmite la relación usuarios del sitio 1 al 2. De acuerdo con los datos de la Fig. 1.6., se necesitan transmitir 1840000 octetos.
3. Calcula la junta de las relaciones que resultan del paso 1 y 2 en el sitio 2; allí queda el resultado.

Si introducimos una operación semijunta, que se representa algebraicamente en la Fig. 2.4. (b), el paso número 2 descrito previamente debe ser remplazado por los pasos siguientes:

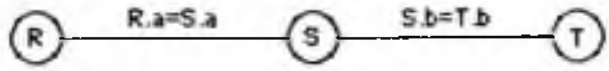
2.1. Calcula la proyección del resultado del paso 1 sobre el campo numusr.
2.2. Transmite el resultado del paso 2.1. al sitio 1. Se trata de 8004 octetos.
2.3. Calcula la semijunta de la relación usuarios con el resultado del paso 2.2.
2.4. Transmite el resultado del paso 2.3. al sitio 2. Se estima que serán 613364 octetos.

Como se observa, el volumen de datos a transmitir en esta segunda estrategia es de sólo 621368 octetos, poco más de la tercera parte del volumen de datos que se transmiten con la primera alternativa. Nótese que, para formar el procedimiento de ejecución adecuadamente, es imperativo que el cálculo de la subexpresión que realiza la selección (paso 1) se haga una sola vez, lo que no se refleja inmediatamente de la expresión relacional de la Fig. 2.4. (b).

Cuando se tiene una consulta en la que aparecen varias juntas que se realizan entre varias relaciones, se usan secuencias de semijuntas para reducir las relaciones antes de transmitirlas; a esta secuencia de semijuntas se le llama programa de semijuntas. Cada semijunta tiene una selectividad asociada, que es el factor en el que va a disminuir la cardinalidad de la relación que se reduce. La selectividad de un programa de semijuntas se calcula en base a la selectividad de las juntas que lo componen.

Una representación conveniente para visualizar los programas de semijunta posibles es la gráfica de consulta, que se forma con un vértice por cada relación involucrada en la consulta, y una arista entre los vértices cuyas relaciones esten involucradas en una operación de junta; los operadores unarios se ignoran y no se considera unión ni producto. La gráfica de consultas se presenta en [8], donde aparece de manera más general la noción de reducción aquí expuesta, y se describe la formación de programas de semijunta que reducen completamente a las relaciones, siempre que la gráfica de consulta sea acíclica. El uso de gráficas acíclicas resulta muy importante en varios areas relacionadas con Bases de Datos [7].

Con ayuda de estas gráficas de consulta, podemos calcular la selectividad de un programa de semijuntas, como se muestra en el siguiente ejemplo.



(a) Una gráfica de consulta.

Semijunta reductora	Selectividad
R SJ S	0.2
S SJ R	0.4
T SJ S	0.6
S SJ T	0.8

(b) Semijuntas simples para la consulta.

Programa de semijuntas	Selectividad
R SJ S SJ T	0.16
R SJ S SJ R	0.2

(c) Algunos programas de semijuntas para la consulta.

Fig. 2.5. Gráficas de consulta y selectividad de programas de semijunta.

Ejemplo 2.4. Tenemos una consulta sobre tres relaciones R, S y T cuya gráfica se muestra en la Fig. 2.5. (a). Con base en esta gráfica, podemos determinar las posibles semijuntas básicas que se muestran en la Fig. 2.5. (b) acompañadas de su selectividad asociada (ésta se calcula de acuerdo con los perfiles de las relaciones). En la Fig. 2.5. (c) aparecen varios programas de semijuntas.
El primer programa de semijuntas pide la reducción de la relación S (con un factor de 0.8) y luego el uso del resultado de esta operación para reducir la relación R. Como la última semijunta se está realizando con una relación reducida, la selectividad total está determinada de acuerdo con la selectividad de las semijuntas involucradas (0.16, por jemplo).
En el segundo programa de semijuntas, por el contrario, la selectividad total permanece como 0.2, ya que la reducción previa de S no representa una selectividad adicional respecto de la relación a reducir R.

3. OPTIMACION BASADA EN REGLAS

En este capítulo se propone la arquitectura de un optimador de consultas basado en reglas. Primero se plantea el problema de la optimación en un marco general; en seguida se hacen algunas consideraciones respecto de la búsqueda de una solución, y por último se presenta la arquitectura del optimador.

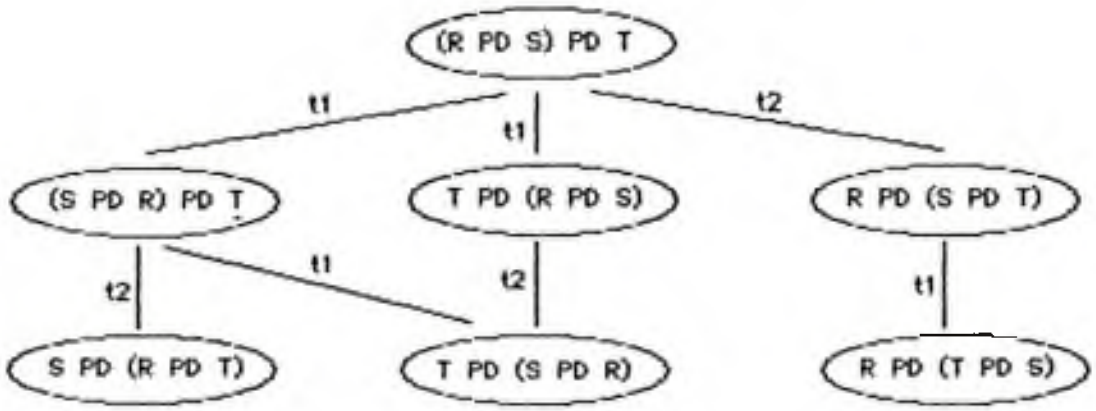
El marco general que se describe en este capítulo está basado en los métodos débiles desarrollados en el área de Inteligencia Artificial [31,35].

3.1. Planteo del problema

Como ya se mencionó (véase 1.1.), el problema de la optimación consiste en generar de un plan de acceso razonablemente "bueno" a partir de una consulta. Consideraremos un plan de acceso como una expresión algebraica extendida. Para resolver este problema se puede usar un enfoque transformacional [12,13,18]: a partir de la consulta se forma un plan de acceso inicial, al que se aplican una serie de transformaciones para llevarnos al plan de acceso solución. Cada una de las transformaciones debe producir un plan de acceso cuyo resultado es el mismo que el del original, pero cuyo tiempo de ejecución puede ser distinto. Este esquema se observa, por ejemplo, en las transformaciones que se muestran en la Fig. 2.3., y se ajusta al planteamiento general que se describe a continuación.

Consideremos un conjunto E de estados y un conjunto T de transformaciones que nos llevan de un estado a otro de E. Señalamos a un estado de E como el estado inicial i, y a un conjunto de estados F (subconjunto de E) como los estados finales. Un problema, descrito en estos términos, consiste en encontrar una secuencia de transformaciones que nos lleven del estado inicial i a cualquier estado final elemento de F. Podemos estar interesados en la secuencia de transformaciones empleada o solamente en el estado final alcanzado. Tenemos además una función f que nos da el costo de cada estado; los estados de F tienen todos el mismo costo, que es mínimo respecto de los costos de los estados de E. En nuestro caso, los estados corresponden a planes de acceso y no nos interesa la secuencia de transformaciones empleada; la función de costo estima el número de páginas de memoria secundaria que es necesario accesar.

El conjunto de estados a los que podemos llegar mediante transformaciones aplicadas sobre el estado inicial constituye el espacio de búsqueda del problema. En nuestro caso, el espacio de búsqueda no está definido explícitamente al comenzar a buscar un plan de acceso, más bien se va generando dinámicamente a medida que aplicamos transformaciones durante la búsqueda. Estos conceptos se ilustran en el siguiente ejemplo.



t1 : Transformación por conmutatividad.
t2: Transformación por asociatividad.

Fig. 3.1. Porción de un espacio de búsqueda.

Ejemplo 3.1. Queremos calcular el producto cartesiano de tres relaciones (R PD S) PD T. En la Fig. 3.1. se muestra una porción del espacio de búsqueda correspondiente, haciendo uso de las transformaciones de asociatividad y conmutatividad del producto cartesiano.

Algunas transformaciones están basadas en propiedades algebraicas de los operadores relacionales, como se observa en el ejemplo anterior, pero no es necesario considerar todas las transformaciones posibles de una propiedad dada. Por ejemplo, sabemos que la relación vacía es un elemento identidad para la unión. De aquí podemos derivar reglas de transformación que simplifican expresiones mediante la eliminación de uniones (R UN VACIA -> R), pero no consideraremos reglas que nos lleven en el sentido inverso, agregando uniones. Las transformaciones que usamos, aunque basadas en propiedades algebraicas, están destinadas a llevarnos sólo a expresiones consideradas como "interesantes".

3.2. Búsqueda de una solución

En esta sección hacemos las consideraciones necesarias para realizar la búsqueda de una solución, de acuerdo con el planteamiento de la sección anterior.

3.2.1. Un algoritmo general de búsqueda

Consideremos un "terreno" formado por estados cuya "altura" está medida por la función de costo f. Nos movemos entre estados adyacentes del terreno mediante la aplicación de

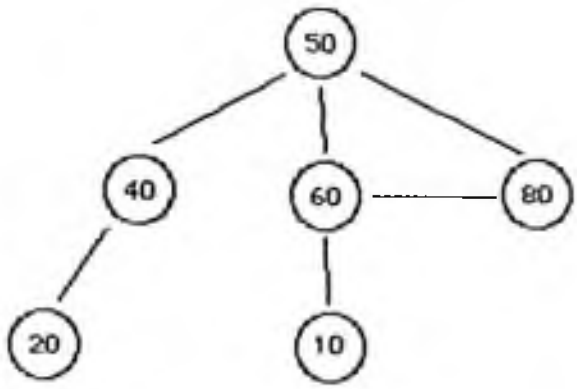
transformaciones, y podemos tener un movimiento descendente, hacia un valle del terreno, o ascendente. El problema consiste en movernos desde un sitio dado hasta el valle más bajo del terreno. El procedimiento consiste en explorar la vecindad del sitio inicial en busca de valles, y elegir el más bajo de ellos. En este recorrido no solo se harán movimientos de descenso; es necesario realizar movimientos de ascenso para dejar un valle y poder explorar otros. Para no examinar todo el terreno, imponemos una restricción a los movimientos de ascenso: con base en el valle mas bajo que hemos encontrado, calculamos una cota para la altura máxima que estamos dispuestos a seguir. El cálculo de ésta se basa en un parámetro que determina qué tan exhaustiva será la búsqueda. Llamaremos al algoritmo de descenso de la colina.

Algoritmo 3.1. Descenso de la colina
Descripción. Localiza un estado con un valor pequeño para f, basado en un factor de ascenso a que dice qué tanto se va a explorar el espacio de búsqueda.

1. Inicializa mejor al estado inicial. Inicializa el conjunto Abierto a que contenga justamente a mejor. Inicializa el conjunto Cerrado como vacío.
2. Si el conjunto Abierto está vacío, termina. El estado solución es mejor.
3. Excluye de Abierto el estado x cuyo valor de f sea mínimo. Incluye x en el conjunto Cerrado.
4. Para cada estado z al que llegamos mediante la aplicación de una transformación sobre x, si z no está en Abierto ni en Cerrado y $f(z) < a * f(mejor)$, inclúyelo en Abierto; y si $f(z) < f(mejor)$, remplaza mejor por z.
5. Continua en el paso 2.

El factor a indica qué tan dispuestos estamos a subir una colina en busca de un valle más bajo que el mejor que hemos hallado ya. En el caso de que a=1, sólo consideraremos expresiones mejores que la que tenemos, produciendo un comportamiento voraz. Por otro lado, si a=infinito, se llevará a cabo una búsqueda exhaustiva en todo el espacio. En el ejemplo siguiente se visualiza el comportamiento de este algoritmo.

Ejemplo 3.2. En la Fig. 3.2. (a) se muestra un espacio de búsqueda pequeño. Cada estado está identificado por su costo y los arcos representan transformaciones posibles.
En la Fig. 3.2. (b) se muestran los valores de las variables mejor, Abierto y Cerrado en cada paso del algoritmo anterior, suponiendo un valor de 1 para el factor a (búsqueda voraz). Se observa que el algoritmo no encuentra el mínimo global.



(a) El espacio de búsqueda.

Mejor	Abierto	Cerrado
50	{50}	{}
40	{40}	{50}
20	{20}	{50,40}
20	{}	{50,40,20}

Solución: 20

(b) Búsqueda con un factor de ascenso de 1.

Mejor	Abierto	Cerrado
50	{50}	{}
40	{40,60}	{50}
20	{20,60}	{50,40}
20	{60}	{50,40,20}
10	{10}	{50,40,20,60}
10	{}	{50,40,60,20,10}

Solución: 10

(c) Búsqueda con un factor de ascenso de 1.25.

Fig. 3.2. Búsqueda con distintos factores de ascenso.

En la Fig. 3.2. (c) se muestran los valores de las variables en cada paso del algoritmo, ahora con un factor de ascenso de 1.25. En caso sí se encuentra el mínimo.
Con ninguno de los dos factores de ascenso anteriores se recorre todo el espacio, así que en ninguno de ellos hay garantía de encontrar la mejor solución. Como se observa, el parámetro a especifica qué tan exhaustiva es la búsqueda.

Una variante de este método es considerar a como una función del tiempo. A medida que transcurre el tiempo el valor de a va disminuyendo, haciendo que los movimientos sean cada vez más restringidos hasta limitarse a un valle. El procedimiento de búsqueda de solidificación (annealing) se basa en un principio semejante al anterior [25].

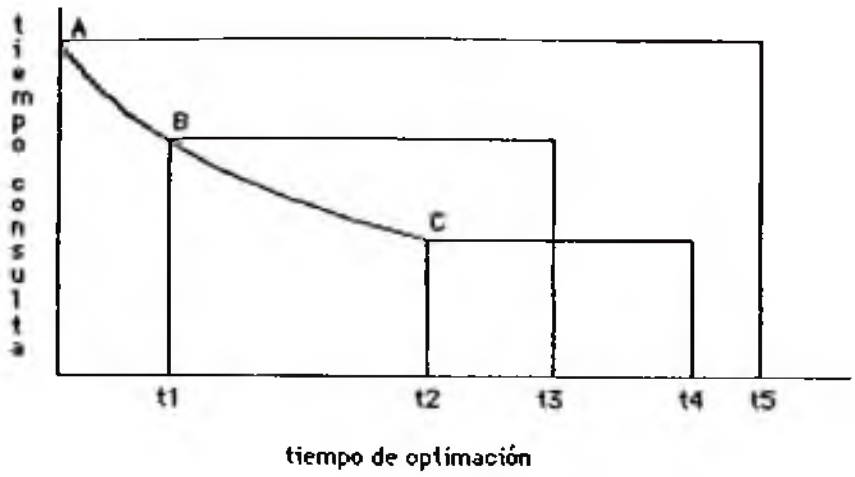


Fig. 3.3. Búsqueda acotada en el tiempo.

3.2.2. Búsqueda acotada en el tiempo

Como el propósito de la optimación de consultas es disminuir el tiempo de respuesta de las mismas, no es aceptable un procedimiento de optimación que tarde más tiempo que la propia consulta. Esto nos lleva a la idea de la búsqueda acotada.

Definimos un factor de optimación o, que dice cuánto tiempo se va a invertir en la optimación de una consulta dada, de acuerdo con su tiempo esperado de ejecución. Independientemente del método de búsqueda que usemos, acotaremos el tiempo de la misma de acuerdo con el factor o. A medida que encontremos mejores expresiones, iremos actualizando la cota, de modo que no se invierta más de la proporción señalada en la optimación de ninguna de las expresiones encontradas. En el siguiente ejemplo se observa esta actualización de cotas.

Ejemplo 3.3. En la Fig. 3.3. aparece una gráfica en la que el eje horizontal representa el tiempo que toma el procedimiento de optimación, y el eje vertical el tiempo estimado de ejecución de la mejor expresión que hemos encontrado en un momento dado; en cada eje se usa una escala distinta. La expresión inicial para la búsqueda es A, cuyo tiempo estimado de ejecución determina el punto del tiempo t5 como cota para el tiempo de optimación de la consulta. En el tiempo t1, encontramos una expresión B con costo menor al de A y que determina una nueva cota t3. Después, en el tiempo t2 se encuentra una mejor expresión C, con un tiempo de inversión en optimación que nos llevaría hasta t4; como la cota actual es menor, nos quedamos con ella. Al llegar al tiempo t3 damos por terminada la búsqueda y reportamos la mejor solución hallada, C.

3.3. Un optimador de consultas basado en reglas

El optimador que se propone recibe como entrada una consulta, en la forma de una expresión algebraica, y genera un plan de acceso para obtener los datos solicitados. Está compuesto de un núcleo en que se incluyen procedimientos generales para la optimación de consultas y varios módulos de configuración, en los que se especifica mediante reglas el tipo de objetos, operadores y métodos a manejar, además de sugerencias en cuanto a cómo llevar a cabo la optimación.

A continuación se definen los elementos que componen el optimador general, así como su arquitectura; en seguida se introduce un optimador configurado para el modelo de base de datos relacional, en el que se ilustrará el papel de cada componente del optimador general.

Se propone un optimador general (OG) basado en reglas que realice una búsqueda acotada en el tiempo usando dos técnicas básicas: el metodo de ascenso de la colina, que corresponde al algoritmo 3.1.; y el método voraz, que está basado en el mismo algoritmo con un factor de ascenso de 1, pero que en el paso 4 elige a la mejor de todas las expresiones que se generan (sólo a una). El optimador tiene un módulo de reconocimiento de subexpresiones comunes, que le sirve para calcular adecuadamente costos y para generar planes de acceso en la forma de programas.

Se define un conjunto de atributos que se asocian tanto a los operandos que pueden usarse en una expresión, como a los resultados intermedios de éstas. Los atributos sirven para almacenar el costo de evaluación y cualquier otra propiedad que nos interese del objeto en cuestión, como su tamaño, cardinalidad o algún parámetro de almacenamiento.

Se define un conjunto de operadores y el tipo de argumentos que pueden tener. Para cada operación definimos un conjunto de métodos de ejecución.

Se define un conjunto de reglas de evaluación, que determinan el valor de los atributos del objeto calculado por una subexpresión, de acuerdo a su construcción.

Con base en las propiedades algebraicas de las operaciones, se define un conjunto de reglas de mejora y otro de reglas de transformación. Las reglas de mejora se aplican sobre las expresiones siempre que ésto es posible, y la búsqueda se realiza mediante la aplicación de las reglas de transformación.

De acuerdo con el tipo de transformaciones necesarias, se incluyen módulos auxiliares, donde se colocan procedimientos de servicio relativos al tipo de objetos manejados.

Con los elementos descritos, la arquitectura del optimador se muestra en la Fig. 3.4. Allí se señala el núcleo del optimador y los módulos de configuración. Mediante estos módulos el optimador se adapta a un modelo determinado, o se extiende en cuanto a operaciones, métodos o reglas de optimación.

A partir de la expresión algebraica de entrada, la máquina de búsqueda se encarga de encontrar, con base en las reglas codificadas por el usuario, una expresión mejor en términos de ejecución, y de ésta se genera un plan de acceso en la forma de

programa. El optimador como componente de un sistema de base de datos se muestra en la Fig. 3.5. La consulta del usuario es puesta primero en la forma de una expresión algebraica, luego el optimador encuentra un plan de acceso para resolverla, y este plan es ejecutado por un módulo de acceso a las estructuras de almacenamiento. El resultado de la consulta es finalmente presentado al usuario por un módulo de reporte de datos.

Por las características del programa, parece conveniente su programación en un lenguaje como Prolog [14], en el que la codificación y uso de reglas es fácil de realizar. Para una implantación exitosa se requiere un sistema de Prolog que sea eficiente en cuanto a tiempo de ejecución, y que proporcione facilidades para llamar a rutinas escritas en otros lenguajes, para hacer posible la integración con el resto del sistema.

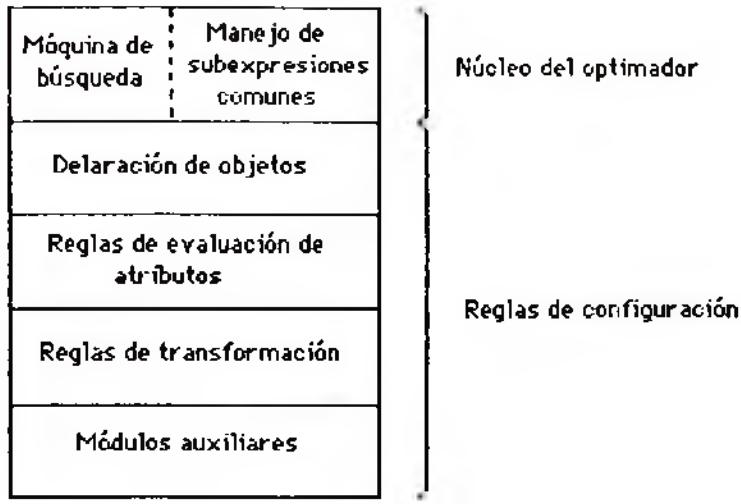


Fig. 3.4. Arquitectura del optimador general.

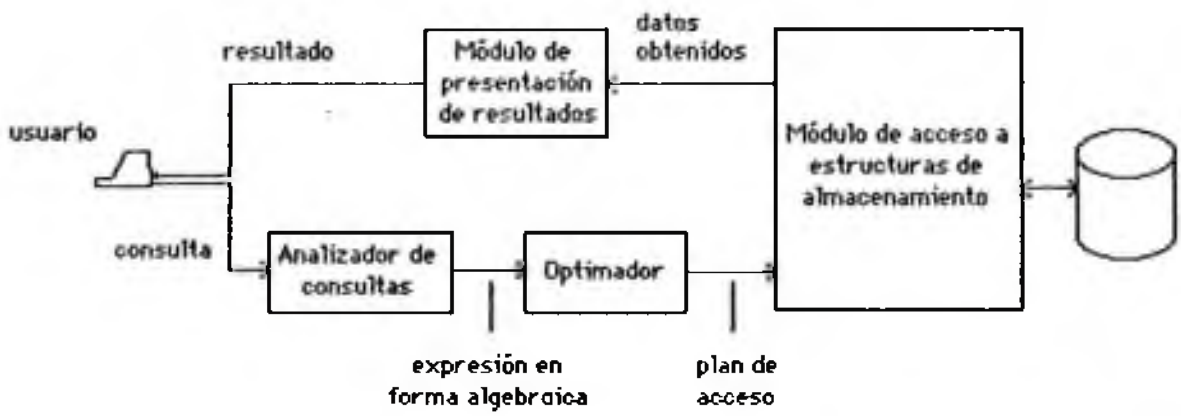


Fig. 3.5. El optimador como componente de un sistema de base de datos.

De acuerdo con la arquitectura general propuesta, se programó un optimador para bases de datos relacionales y distribuidas (ORD). En la Fig. 3.6. se muestra un esquema de los módulos del mismo, que se describen en detalle en el capítulo siguiente. Se desarrolló también un sistema de prueba para el optimador, cuyo esquema aparece en la Fig. 3.7. El usuario formula consultas que son convertidas a una forma algebraica. El optimador realiza su tarea y escribe en una bitácora, en memoria secundaria, las consultas que ha procesado y los planes de acceso que ha generado para ellas; luego presenta al usuario la expresión encontrada como solución.

El optimador está programado en Turbo-Prolog [10]. Aunque éste se desvía del Prolog estándar, tiene varias características importantes: es un compilador que genera programas relativamente eficientes en cuanto a tiempo de ejecución, comparado con otras implantaciones del lenguaje; proporciona números reales y funciones trascendentales, necesarios para la estimación de costos; permite la llamada a rutinas escritas en otros lenguajes; y excluyendo algunos predicados cuyo uso se puede evitar, es un subconjunto del Prolog estándar y puede ser traducido fácilmente a este último.

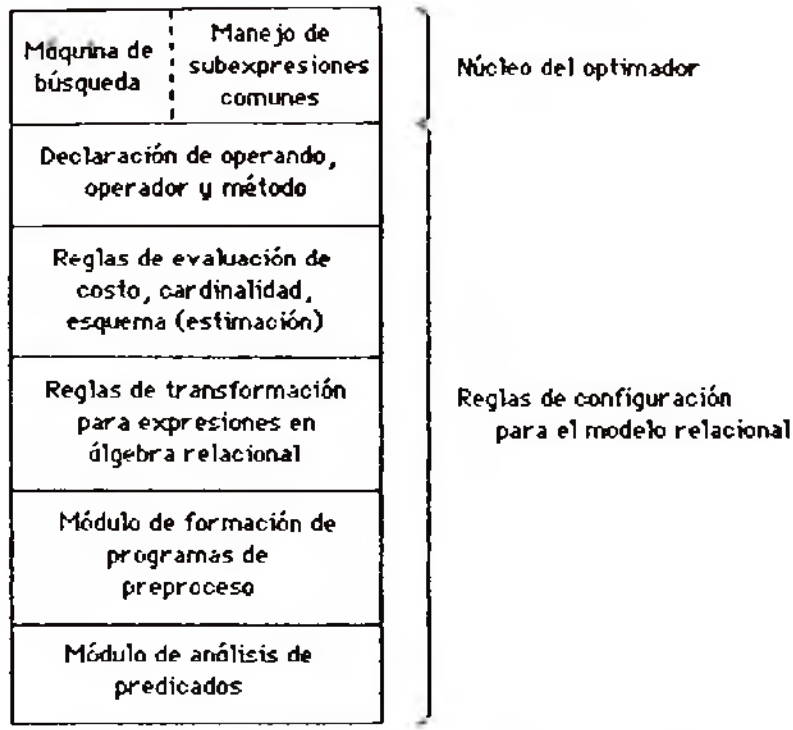


Fig. 3.6. Módulos de un optimador para base de datos relacionales y distribuidas.

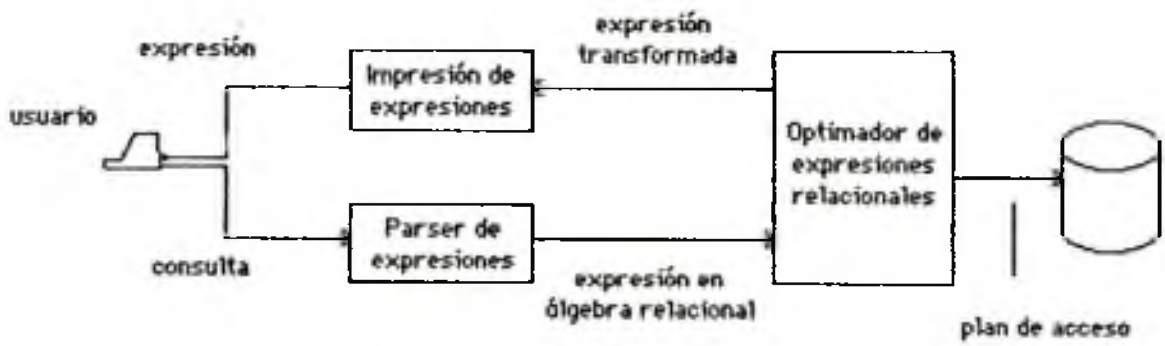


Fig. 3.7. Sistema de prueba para el ORD.

4. UN OPTIMADOR PARA EL MODELO RELACIONAL

En este capítulo se describe en detalle el optimador para base de datos relacionales y distribuidas introducido en la sección 1.4, que sigue la arquitectura y convenciones del optimador general basado en reglas (OG).

Se describen los módulos del optimador, de acuerdo el esquema de la Fig. 3.6., mostrando una aplicación concreta de los elementos definidos en el OG.

4.1. Operandos y operadores

El tipo de operando que se maneja en el modelo relacional, la relación, se describe mediante un registro con los siguientes componentes, que forman un perfil de la relación [13]:

- Nombre de la relación.
- Esquema de la relación. El esquema de una relación es una lista de campos, donde cada campo se forma por un nombre de campo, ancho del campo en octetos y número de valores distintos (val) que tiene en las eneadas de la relación.
- Cualidad de la relación. Condiciones que corresponden a las restricciones de integridad de la relación.
- Cardinalidad de la relación.
- Ancho de la relación, en octetos que ocupa cada eneada.
- Sitios en los que está almacenada la relación.
- Nombre del campo por el cuál está clasificada la relación, si se aplica.
- Nombres de los campos sobre los cuales se tiene índices para accesar la relación.

Los nombres de campos se componen de un prefijo y una raíz, separados por un punto. El prefijo suele ser el nombre de la relación a la que pertenecen. Los nombres de campos de todas las relaciones de la base de datos deben ser distintos.

Los operadores que se manejan en el ORD son los descritos en la sección 1.2.2., y se listan a continuación:

- Selección. Usa una lista de condiciones separadas por el conectivo and. Cada condición es una comparación, que puede usar los operadores $<$, $=$ y $>$, entre dos elementos que pueden ser el nombre de un campo o una constante.
- Proyección. Usa una lista de nombres de campos sobre los que se realiza la proyección.
- Unión. Como en la unión se requiere que los nombres de campos de las relaciones sobre las que opera sean iguales, y los nombres de campos de todas las relaciones de la base de datos son distintos, se usa un renombramiento de prefijos. Este renombramiento cambia los nombres de campos de las relaciones sustituyendo el prefijo de las mismas. Después del renombramiento los nombres de campos de las relaciones sobre las que se opera deben ser iguales.

- Diferencia. Al igual que la unión, esta operación hace uso de un renombramiento de prefijos para que sean iguales los nombres de campos de las relaciones sobre las que opera.
- Producto cartesiano.
- Junta. Usa una lista de condiciones como la empleada en una selección.
- Semijunta. Usa, también, una lista de condiciones como la empleada en la junta.

Se tiene un conjunto de atributos asociado a cada subexpresión de una expresión. Los atributos manejados en el ORD se listan en seguida:

- Esquema de la relación resultante, semejante al esquema de un operador como se describió anteriormente.
- Cualidad de la relación, como la que ya se describió.
- Las Reducciones que ha sufrido la relación. Estas reducciones dependen de las juntas o semijuntas de las que ha sido objeto. Se codifican en la forma de una lista de condiciones junto con los esquemas de las relaciones involucradas. Este atributo sirve para determinar si es aplicable una operación de semijunta y, en caso de que lo sea, para calcular su selectividad.
- La cardinalidad de la relación resultante.
- El ancho de la relación resultante.
- El sitio en se queda la relación resultante.
- El nombre del campo por el cual esta clasificada la relación resultante.
- Los nombres de los campos sobre los cuales se tienen índices.
  El lugar en que se encuentra la relación [30]. Se puede encontrar en disco o en secuencia. Si está en disco, se encuentra almacenada y es necesario hacer accesos a memoria secundaria para manejarla; si está en secuencia, las eneadas son el resultado de una operación que las "produce" una a una, y de este modo pueden ser tomadas por otra operación sin la necesidas de hacer ningún acceso a memoria secundaria.
- El costo de la operación principal de la subexpresión (la última que se realiza), medido en número de páginas de memoria secundaria que se necesitan leer o escribir para ejecutar la operación.

A continuación se muestran los métodos considerados para la ejecución de cada uno de los operadores del álgebra relacional. Cada método tiene asociada una fórmula para estimar su costo de ejecución, medido en número de páginas de memoria secundaria que necesita accesar. Para su ejecución, cada método espera que las relaciones argumento cumplan con ciertas características (como índices, por ejemplo); si una relación argumento no satisface estas características, debe ser preprocesada, como se verá más adelante. El costo total de ejecución para una operación debe tener en cuenta el costo de preproceso.

Selección.

- Simple. Considera, una por una, las eneadas de la relación argumento y conserva sólo las que cumplen con las condiciones de la selección. En este, como en el método siguiente, el resultado queda en la forma de secuencia descrita previamente. El costo de la operación es cero, pero la relación argumento debe estar en forma de secuencia.
- Indice. Elige una de las condiciones de la selección y utiliza un índice para accesar rápidamente las eneadas que cumplen con ella. Aplica luego el resto de las condiciones sobre estas eneadas y conserva sólo las que las satisfacen. Se espera accesar tantas páginas como eneadas tenga la relación resultante; la expresión argumento debe estar en disco con el índice apropiado.

Proyección.

- Simple. Considera, una por una, las eneadas de la relación argumento y almacena en disco la proyección solicitada. Clasifica luego el resultado para eliminar eneadas que hayan resultado duplicadas. La relación argumento debe estar en secuencia; el resultado de esta operación queda en disco y su costo es el número de páginas que vaya a ocupar la relación resultante. El costo de clasificación se ve más adelante.

Unión.

- Simple. Considera las eneadas de cada una de las relaciones argumento y almacénalas en disco. Clasifica la relación resultante para eliminar duplicados. Las relaciones argumento deben estar en secuencia; el costo corresponde al número de páginas necesarias para almacenar el resultado, que queda en disco, además del costo de clasificación.

Producto.

- Ciclo. Toma una relación como la relación externa y, para cada una de sus eneadas, recorre secuencialmente la otra relación (interna) para generar cada eneada del resultado. En los métodos de ejecución de producto y junta, la relación que queda del lado izquierdo en la expresión algebraica correspondiente se toma como externa y el resultado queda en la forma de secuencia; la relación externa debe estar en secuencia y la interna en disco, el costo es el producto de las páginas de la relación externa por las de la interna.

Junta.

- Ciclo. Procede como en el caso de un producto, pero elimina las eneadas que no satisfagan las condiciones de la junta; los requisitos y costos son los mismos que en el producto.
- Indice. Para cada una de las eneadas de la relación externa, usa un índice sobre la relación interna para accesar las eneadas que cunplen con la condición de junta.

La relación interna debe tener el índice apropiado, y el costo es el número de eneadas de la relación externa.
- Mezcla. Dada una condición de igualdad, recorre cada una de las relaciones de acuerdo con la clasificación del campo involucrado, genera las eneadas para las que se satisfaga la condición. Tanto la relación externa como la interna deben estar en secuencia y clasificadas apropiadamente; el costo es cero.

Semijunta.
- Ciclo. En este, como en los métodos siguientes, de ejecución de semijunta, se toma la relación a reducir (la izquierda) como relación externa y el resultado queda en forma de secuencia. Se conserva una eneada si es que existe la eneada correspondiente en la relación interna que satisface la condición. En este método se sigue una estrategia semejante a la de Junta-Ciclo, y su costo se estima del mismo modo.
- Indice. Semejante a la estrategia Junta-Indice.
- Mezcla. Semejante a la estrategia Junta-Mezcla.

Los métodos para unión, producto y junta llevan, además, un parámetro numérico que indica la estrategia de transmisión: si el parámetro es cero, transmite la relación izquierda al sitio de la relación derecha y allí se realiza la operación; si es distinto de cero, indica el número de sitio donde se van a transmitir ambas relaciones para luego realizar allí la operación binaria.

Antes de que se ejecute una operación, las relaciones argumento son llevadas a la forma requerida mediante un programa de preproceso que, dependiendo del formato actual de una relación y el formato que requiere el método, incluye las operaciones necesarias para hacer la conversión. A continuación se listan las operaciones de preproceso consideradas:

- Almacena. Toma una relación que está en forma de secuencia y déjala en disco. Su costo es el número de páginas que ocupa la relación.
- Recorre(C). Recorre una relación de acuerdo con el índice del campo C y déjala en forma de secuencia. Si C se omite, recórrela secuencialmente. Su costo es el número de páginas que ocupa la relación.
- Indice(C). Esta operación se aplica sobre relaciones en disco, y lo que hace es construir un índice sobre el campo C. Si la relación ocupa x páginas, el costo de construir el índice se estima como $x * \ln(x)$.
- Clasifica(C). Clasifica una relación en disco de acuerdo al campo (C). El costo estimado es el mismo que el de construir un índice.
- Envía(S). Toma una relación en secuencia y envíala al sitio S. El costo se calcula como el número de páginas a trnsmitir multiplicado por un factor de transmisión, que indica la proporción en que es más cara la transmisión de datos que el acceso a memoria secundaria. La relación queda almacenada en el sitio de destino.

Se tiene un parámetro de tiempo de acceso a memoria secundaria, que sirve para estimar el tiempo de ejecución de una consulta con base en su costo. Esto se usa para llevar a cabo la búsqueda acotada en tiempo.

En consultas distribuidas se tiene un sitio de usuario, donde debe aparecer finalmente el resultado de la consulta. Si este resultado queda en un sitio distinto al del usuario, es necesaria su transmisión final al sitio correcto.

4.2. Reglas de evaluación de atributos

Para cada operador y método es necesario codificar una serie de reglas de evaluación de atributos, que sirvan para calcular los atributos de la relación resultante de la aplicación de una operación mediante un cierto método. Los atributos cualidad, reducción, ancho, sitio, clasificación, índices, lugar y costo son relativamente sencillos de determinar a partir de la explicación de cada método dada en la sección anterior.

Para la estimación de la cardinalidad y esquema de la relación resultante utilizamos las estimaciones descritas en [13]. Supondremos que los campos de las relaciones son independientes en cuanto a la distribución probabilística de sus valores. Los procedimientos de estimación se describen a continuación.

En el caso de una selección, asociamos un factor de selectividad a la condición usada. Si la condición se trata de una igualdad entre el campo C y una constante, la selectividad se calcula como el inverso del val(C); si se trata de una igualdad entre dos campos C y D, la selectividad es el mínimo de los inversos de val(C) y val(D); en otro caso se considera una selectividad de 0.5. La cardinalidad de la relación resultante es el factor de selectividad multiplicado por la cardinalidad de la relación argumento.

El val de los campos de la relación resultante se calcula con base en una estimación propuesta en [36]: sea m el val inicial del campo y r la cardinalidad de la relación resultante; el nuevo val del campo se calcula como:

$$v(m,r) = \begin{cases} r & \text{si } r < m/2 \\ (r+m)/3 & \text{si } m/2 \leq r < 2m \\ m & \text{si } 2m \leq r \end{cases}$$

Una excepción a la estimación anterior se da cuando un campo aparece en una condición de igualdad comparado contra una constante; en este caso, el val del campo es 1.

En una proyección es sencillo calcular el esquema de la relación resultante; los campos conservan su val. Se calcula el producto del val de los campos proyectados y, si es menor a la cardinalidad de la relación original, la nueva cardinalidad es este producto; en otro caso, se conserva también la cardinalidad.

Para la unión, estimamos la cardinalidad como la suma de las cardinalidades de los argumentos, y el val se trata de manera semejante.

En el producto cartesiano, la cardinalidad del resultado es el producto de las cardinalidades de los argumentos, mientras que el val se conserva para todos los campos.

Para una junta, encontramos las relaciones reducidas (como si se hubieran tratado mediante semijuntas). Estas relaciones reducidas sólo conservan las eneadas que van a participarn en la junta. Si la condición involucra al campo a, la cardinalidad del resultado se estima como el producto de las cardinalidades de las relaciones reducidas, dividido entre el val de a en una relación reducida. Puede tomarse cualquiera de los campos involucrados en la condición mediante una comparación de igualdad. El val de los campos del resultado se toma de las relaciones reducidas.

Para una semijunta, se calcula primero una selectividad asociada (su cálculo se verá en 4.5.2.), que se aplica sobre la cardinalidad de la operación a reducir de manera similar a como se hace en una selección. El val del campo involucrado en la condición de semijunta disminuye con el mismo factor, y para el resto de los campos se emplea la aproximación descrita para la selección.

4.3. Reglas de transformación de expresiones

Basados en las propiedades algebraicas de los operadores relacionales, en el ORD se incluyen reglas de transformación para llevar a cabo la búsqueda de acuerdo con el esquema del OG. A continuación se presentan las reglas de mejora y de transformación consideradas.

4.3.1. Reglas de mejora

Con base en las propiedades algebraicas, encontramos ciertas transformaciones que siempre mejoran el tiempo de ejecución de una consulta. Estas transformaciones son aplicadas por el ORD sobre cualquier subexpresión de una expresión, siempre que la aplicación sea posible. En la Fig. 4.1. se muestran las reglas de mejora utilizadas. Las variables F, F1, .. representan condiciones; C, C1, .. representan conjuntos de campos; y E, E1, .. representan expresiones. atr determina los campos de una relación o condición; cnt determina si un cierto predicado es contradictorio; nrm normaliza un predicado; y cld determina la cualidad de una relación. A continuación se describen las reglas, con referencia a la Fig. 4.1.

SL(F):PJ(C):E —> PJ(C):SL(F):E

(a) Aplica las selecciones antes de las proyecciones.

SL(F1):SL(F2):E —> SL(F1 and F2):E
PJ(C1):PJ(C2):E —> PJ(C1):E
{ C1 ⊆ C2 }

(b) Combina dos operadores unarios en uno solo.

SL(F):PD:E1,E2 --> JN(F):E1,E2
SL(F1):JN(F2):E1,E2 --> JN(F1 and F2):E1,E2

(c) Usa la operación de junta siempre que se pueda.

SL():E --> E
PJ(atr(E)):E --> E
JN():E1,E2 --> PD:E1,E2

(d) Simplifica operaciones innecesarias.

SL(F):UN:E1,E2 --> UN:SL(F):E1,SL(F):E2
PJ(C):PD:E1,E2 --> PD:PJ(C1):E1,PJ(C2):E2
{ C1 = C ∩ atr(E1), C2 = C ∩ atr(E2) }
PJ(C):JN(F):E1,E2 --> JN(F):PJ(C1):E1,PJ(C2):E2
{ C1 = (C U atr(F)) ∩ atr(E1), C2 = (C U atr(F)) ∩ atr(E2)}
PJ(C):UN:E1,E2 --> UN:PJ(C):E1,PJ(C):E2
JN(F1 and F2 and F3):E1,E2 --> JN(F3):SL(F1):E1,SL(F2):E2
{ atr(F1) ⊆ atr(E1), atr(F2) ⊆ atr(E2) }

(e) Aplica operadores unarios tan pronto como sea posible.

SL(F):E1 --> VACIA
{ cnt(F and cld(E1)) }
JN(F):E1,E2 --> VACIA
{ cnt(F and cld(E1) and cld(E2)) }
SL(F1):E1 --> SL(F2):E1
{ F2 = nrm(F1, cld(E1)) }
JN(F1):E1,E2 --> JN(F2):E1,E2
{ F2 = nrm(F1, cld(E1) and cld(E2) }

(f) Detecta relaciones vacías y normaliza predicados.

SL(F):VACIA --> VACIA
PJ(C):VACIA --> VACIA
PD:E,VACIA --> VACIA
JN(F):E,VACIA --> VACIA
UN:E,VACIA --> VACIA

(g) Simplificaciones con la relación vacía.

Fig. 4.1. Reglas de mejora.

Se usa la regla (a) porque el resultado de una selección puede tomarse de memoria principal para aplicarle en seguida la proyección.

Las reglas (b) eliminan proyecciones redundantes y combinan dos selecciones en una sola. Las reglas (c) reconocen el operador compuesto de junta, que tiene métodos de ejecución eficientes.

Las reglas (d) eliminan, de acuerdo con el operador, una selección o junta sin condiciones, o una proyección que se hace sobre todos los campos de una relación. Aunque es difícil que un usuario maneje los operadores en este modo, estas configuraciones pueden aparecer en las expresiones que se generan durante la búsqueda.

Las reglas (e) tienen el propósito de reducir los resultados intermedios de la expresión. Esta transformación se va a aplicar si las relaciones argumento están en sitios distintos, en cuyo caso hay necesidad de realizar una transmisión y es necesario reducir las relaciones tanto como sea posible. También se aplica si las relaciones argumento no tienen índices; si se usara la transformación en este caso, se eliminaría la posibilidad de usar los índices para la ejecución posterior de una junta.

Las reglas (f) corresponden a transformaciones basadas en el módulo de análisis de predicados (véase 4.5.1.). Remplaza subexpresiones por la relación vacía si encuentra condiciones contradictorias, o simplemente normaliza las condiciones.

Las reglas (g) se encargan de las simplificaciones que se realizan con la relación vacía. Típicamente se aplicarán estas reglas después del descubrimiento de una relación vacía por las reglas (f).

4.3.2. Reglas de transformación

Se presentan ahora una serie de transformaciones que, aunque no garantizan una reducción en el tiempo de ejecución de las consultas, sí pueden conducir a él. Estas son las transformaciones que se aplican para realizar la búsqueda propiamente dicha de una expresión solución. En la Fig. 4.2. se muestran las reglas de transformación utilizadas. Se usan las mismas variables y funciones que en 4.3.1., además de las siguientes: las variables S, S1, .. representan sitios; OP, OP1, .. representan operadores; MET, MET1, .. representan métodos; ARG, ARG1, .. representan argumentos de operadores; y REL representa una relación. met determina el conjunto de métodos de un operador; can determina los sitios candidatos para realizar una operación binaria; y sit determina los sitios en que está almacenada una relación. A continuación se explican estas reglas, con referencia a la Fig. 4.2.

Las reglas (a), (b) y (c) realizan cambios en el orden de las relaciones de operadores binarios y en el orden de aplicación de los propios operadores. En base de datos distribuidas, estos cambios son necesarios para la formación de programas de semijuntas.

PD :E1 ,E2 --> PD :E2 ,E1
JN(F) :E1 ,E2 --> JN(F) :E2 ,E1
UN :E1 ,E2 --> UN :E2 ,E1

(a) Conmutatividad de operadores binarios.

PD :PD :E1 ,E2 ,E3 <--> PD :E1 ,PD :E2 ,E3
JN(F1 and F2) :JN(F3) :E1 ,E2 ,E3 <--> JN(F1 and F3) :E1 ,JN(F2) :E2 ,E3
{ atr(F1) ⊆ atr(E1) ∪ atr(E3), atr(F2) ⊆ atr(E2) ∪ atr(E3), atr(F3) ⊆ atr(E1) ∪ atr(E2) }
UN :UN :E1 ,E2 ,E3 <--> UN :E1 ,UN :E2 ,E3

(b) Asociatividad de operadores binarios.

JN(F) :E1 ,UN :E2 ,E3 <--> UN :JN(F) :E1 ,E2 ,JN(F) :E1 ,E3
PD :E1 ,UN :E2 ,E3 <--> UN :PD :E1 ,E2 ,PD :E1 ,E3

(c) Distributividad de operadores binarios.

JN(F1 and F2) :E1 ,E2 --> JN(F1 and F2) :SJ(F1) :E1 ,E2 ,E2

(d) Introducción de una semijunta.

SL(F) :UN :E1 ,E2 --> UN :SL(F) :E1 ,SL(F) :E2
PJ(C) :PD :E1 ,E2 --> PD :PJ(C1) :E1 ,PJ(C2) :E2
{ C1 = C ∩ atr(E1), C2 = C ∩ atr(E2) }
PJ(C) :JN(F) :E1 ,E2 --> JN(F) :PJ(C1) :E1 ,PJ(C2) :E2
{ C1 = (C ∪ atr(F)) ∩ atr(E1), C2 = (C ∪ atr(F)) ∩ atr(E2) }
PJ(C) :UN :E1 ,E2 --> UN :PJ(C) :E1 ,PJ(C) :E2
JN(F1 and F2 and F3) :E1 ,E2 --> JN(F3) :SL(F1) :E1 ,SL(F2) :E2
{ atr(F1) ⊆ atr(E1), atr(F2) ⊆ atr(E2) }

(e) Aplica operadores unarios tan pronto como sea posible.

OP[MET1](ARG) :E --> OP[MET2](ARG) :E
{ MET2 ∈ met(OP) }
OP[MET1](ARG) :E1 ,E2 --> OP[MET2](ARG) :E1 ,E2
{ MET2 ∈ met(OP) }

(f) Cambia el método de ejecución de un operador.

OP[MET S1](ARG) :E1 ,E2 --> OP[MET S2](ARG) :E1 ,E2
{ S2 ∈ can(S1) }

(g) Cambia la estrategia de transmisión para un operador binario.

REL(R,S1) --> REL(R,S2)
{ S2 ∈ sit(R) }

(h) Elige otro sitio de donde tomar una relación.

Fig. 4.2. Reglas de transformación.

La regla (d) introduce una semijuntas para la reducción de relaciones que forman parte de una junta. Para introducir una semijunta es necesario que las relaciones involucradas en la junta se encuentren en sitios distintos, y que la operación de semijunta incluya nuevas condiciones en la reducción de la relación que se quiere reducir.

Las reglas (e) hacen uso de la distributividad de operadores unarios respecto de binarios, igual que las reglas (e) de 4.3.1., pero estas reglas se aplican aun cuando se destruyan índices.

Las reglas (f) cambian el método de ejecución de un operador, mientras que las reglas (g) cambian la estrategia de transmisiones para la realización de una operación binaria. Los sitios considerados como candidatos para recibir las relaciones y luego realizar la operación binaria son el sitio de usuario (donde se solicita el resultado), y cualquiera de los sitios en que esté almacenada una relación que se tenga que operar con nuestro resultado.

La regla (h), por último, elige un sitio distinto de donde tomar una relación, si es que ésta se encuentra almacenada en varios sitios.

4.4. Formación de programas de preproceso

El módulo de formación de programas de preproceso es usado para calcular el costo total de evaluación de las operaciones y para formar el plan de ejecución en forma de programa. Para usarlo, se le mandan los atributos de una relación intermedia y un formato de salida que nos interesa, en términos de atributos; el módulo genera un programa de preproceso, una secuencia de operaciones que hay que aplicar sobre la relación para que sus atributos cumplan con lo solicitado. También se calcula el costo del programa de preproceso. Aunque el planteo del problema es muy general (semejante al que aparece en [20]), las secuencias de operaciones que se requieren no son complicadas, y el módulo es relativamente sencillo.

Un servicio adicional que proporciona este módulo es la selección de una condición para la ejecución de selecciones y juntas con base en índices. A partir de los campos de una relación sobre la que se aplicarán ciertas condiciones, elige un condición considerando el tipo de comparación que se usa y los índices que ya tiene la relación.

4.5. Análisis de predicados

Este es un módulo de apoyo que sirve para normalizar y detectar contradicciones con el tipo de condiciones que se manejan en el ORD. Este módulo también realiza la tarea del cálculo de la selectividad de una semijunta, tomando en cuenta las reducciones previas de las relaciones argumento.

4.5.1. Modelación de predicados

Aun tratándose del tipo simple de condiciones que manejamos, podemos encontrar muchas condiciones equivalentes a una dada. Por este motivo, vamos a proponer una modelación de condiciones que nos permita normalizar la representación. Se propone el uso de un gráfica de modelación, que permite además detectar contradicciones.

Algoritmo 4.1. Modelación de una condición.
Descripción. Se construye una gráfica dirigida que modela una condición.
Método.

1. Para cada elemento (campo o constante) presente en la condición que se quiere modelar, forma un conjunto que lo contenga exactamente.
2. Para cada comparación de igualdad presente en la condición, encuentra los conjuntos que contienen a los elementos que se comparan, y remplaza estos conjuntos por su unión. Con esto se construyen clases de equivalencia que reflajan el uso de la relación $=$ en la condición.
3. Construye un vértice de una gráfica por cada conjunto que haya quedado después del paso 2.
4. Por cada comparación de desigualdad, encuentra los vértices asociados a los conjuntos que contienen los elementos que se comparan. Agrega una arista que vaya del vértice asociado al elemento mayor al vértice del elemento menor.
5. Para cada pareja de vértices en los que se encuentren elementos constantes, agrega una arista que refleje el orden entre ellos.

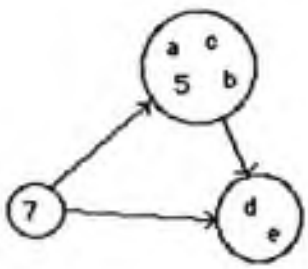
Si más de una constante aparece en uno de los vértices de la gráfica, detectamos un primer caso de contradicción; el segundo caso es la presencia de un ciclo en la gráfica. Con este modelo es sencillo probar si se implica una cierta condición, verificando si dos elementos se encuentran en la misma clase de equivalencia o si existe una trayectoria entre dos vértices de la gráfica.

Si construimos los modelos para dos condiciones distintas y la cerradura transitiva de las gráficas correspondientes nos da gráficas iguales, las condiciones son equivalentes.

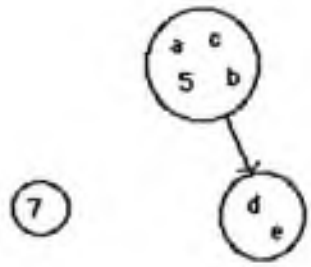
Dada una gráfica de modelación, podemos encontrar una gráfica con menos aristas que la original, pero cuya cerradura transitiva es la misma. A esta nueva gráfica la llamaremos gráfica de modelación normal. El procedimiento para obtenerla se describe a continuación.

(a=b)(b=c)(a>d)(c=5)(e=d)(a<7)(e<7)

(a) Predicado inicial.



(b) Modelo del predicado.



(c) Modelo normal.

(a=5)(b=5)(c=5)(d<5)(d=e)

(d) Predicado normal.

Fig. 4.3. Modelación y normalización de predicados.

Algoritmo 4.2. Normalización de una gráfica.
Descripción. Dada una gráfica de entrada, construida mediante el algoritmo 4.1., construimos su gráfica normal correspondiente. La gráfica normal tiene el número mínimo de aristas posible tal que después de aplicarle el paso el 5 del algoritmo 4.1., su cerradura transitiva es igual a la de la gráfica de entrada.
Método.
1. Para todas las aristas a que conecten dos vértices v1 y v2, si existe una trayectoria de v1 a v2 con longitud mayor que uno, elimina la arista a.
2. Para todas las aristas a que conecten dos vértices v1 y v2, si en las clases de equivalencia de ambos vértices aparecen elementos constantes, elimina la arista a.

A partir de la gráfica normal calculada con el algoritmo anterior, se construye una nueva condición, que llamaremos condición normal; en su construcción, se da preferencia a una comparación contra constante más que contra otro campo. Este procedimiento se ilustra en el siguiente ejemplo.

Ejemplo 4.1. Queremos normalizar las condiciones
(a=b) and (b=c) and (a>d) and (c=5)
and (e=d) and (a<7) and (e<7)
Esta condición inicial aparece en la Fig. 4.3. (a). En la Fig. 4.3. (b) se muestra la gráfica que sirve de modelo al predicado; como se observa, no es un predicado contradictorio. En la Fig. 4.3. (c) aparece la gráfica normal correspondiente; en (d), el predicado normal.

El procedimiento que se sigue para normalizar la condición de una selección o junta es el siguiente: Se forma un modelo de la condición de selección junto con la cualidad de la relación sobre la que se aplica; si se detecta una contradicción, el resultado de la operación es vacío; de otro modo, se obtiene una

condición normal del modelo y se eliminan de ésta todas las comparaciones que se implican como válidas de la cualidad que ya tiene la relación.

4.5.2. Cálculo de selectividad para programas de semijuntas

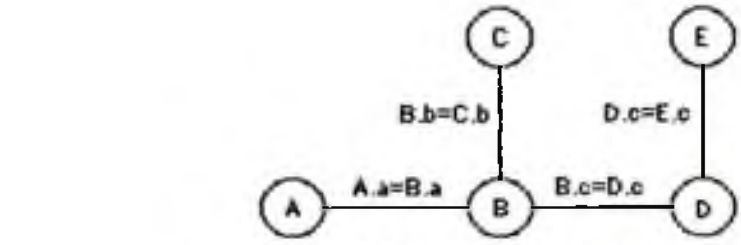
Para estimar los efectos de una operación de semijunta, es necesario calcular su factor de selectividad . Para ésto pueden usarse gráficas de consulta. En el ORD, se hace una estimación de la selectividad de manera semejante a como se realiza en [9]. Supondremos que los campos de las relaciones son independientes en cuanto a la distribución probabilística de sus valores.

En [8] se exponen las dificultades en el uso de semijuntas como estrategia de reducción cuando la gráfica de consulta es cíclica. En estos casos hay que realizar alguna operación que transforme la gráfica en acíclica; una descripción completa de los procedimientos usados para este propósito se puede encontrar en [27]. El procesamiento de la consulta, en estos casos, es primero romper el ciclo y después resolver la gráfica acíclica resultante. Como el ORD no realiza un procesamiento "por etapas", este paso inicial de ruptura de ciclos no encaja naturalmente en el planteo y, por simplicidad, no está incluido.

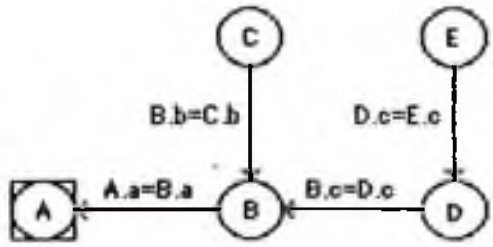
Sólo consideraremos reducciones en las que la gráfica de consulta correspondiente sea acíclica. Dada una reducción, se construye una gráfica de consulta con las condiciones que incluye la reducción y las relaciones involucradas; siendo esta gráfica no dirigida, acíclica y conectada, resulta ser un árbol. Señalamos la relación que se quiere reducir como la ráiz del árbol, y determinamos un programa de semijuntas que la reduce.

> Algoritmo 4.3. Cálculo de una relación reducida.
> Descripción. Dado un árbol que corresponde a una gráfica de consulta en el que la relación a reducir está colocada en la raíz, determina cuál será la relación reducida después de la aplicación del programa de semijuntas reductor.
> Método.
> 1. Si el único vértice de la gráfica es la raíz del árbol, termina. La relación ya está reducida.
> 2. Para cada uno de los hijos de la raíz del árbol, calcula su reducción colocándolo como la raíz del árbol formado por todos sus descendientes.
> 3. Para cada vértice v hijo de la ráiz (ya reducido), actualiza la relación raíz calculando su semijunta con v. Si es a el campo de v que está involucrado en la condición de semijunta, considera la selectividad como $val(a)/dom(a)$; donde $dom(a)$ es el número de valores distintos que puede tomar cualquier campo comparable con a (el dominio del campo).

El algoritmo anterior determina una secuencia de semijuntas que conducen a la reducción de la relación que nos interesa. Los programas de semijunta se calculan independientemente para cada relación, y no es raro que una misma semijunta aparezca en más de un programa reductor; el módulo de reconocimiento de

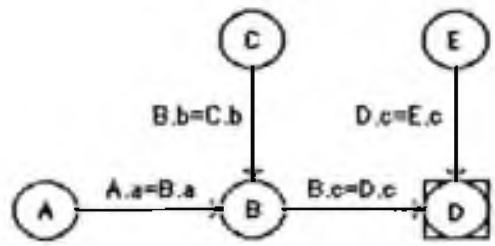


(a) Gráfica de consulta.



Relación a reducir: A
Programa de semijuntas:
D' = D SJ E
B' = B SJ D'
B'' = B' SJ C
A' = A SJ B''

(b) Reducción de A mediante un programa de semijuntas.



Relación a reducir: D
Programa de semijuntas:
B' = B SJ A
B'' = B' SJ C
D' = D SJ B''
D'' = D' SJ E

(c) Reducción de D mediante un programa de semijuntas.

Fig. 4.4. Reducción de relaciones mediante programas de semijuntas.

subexpresiones comunes considera sólo una vez a cada semijunta repetida para el calculo de costos y la formación del planes de acceso en la forma de programa.

En el ejemplo siguiente se ilustran los programas de semijuntas obtenidos mediante el algoritmo anterior.

Ejemplo 4.2. Tenemos las relaciones A, B, C, D y E. La condición de junta que tenemos es

(A.a=B.a) and (B.b=C.b) and (B.c=D.c) and (D.c=E.c)

En la Fig. 4.4. (a) se muestra la gráfica de consulta correspondiente. Usando el algoritmo anterior, encontramos el programa de semijuntas para la reducción de la relación A, como se muestra en la Fig. 4.4. (b), donde A aparece encerrada en un cuadro. En la Fig. 4.4. (c) se muestra el programa de semijuntas para la reducción de D.

La selectividad de una reducción, respecto de una relación, es la cardinalidad de la relación reducida obtenida por el algoritmo anterior dividida entre la cardinalidad de la relación inicial. La selectividad de una semijunta se calcula como la

selectividad de la reducción que se tiene después de la junta dividida entre la selectividad de la reducción inicial.

Cuando en una consulta no se solicitan los campos de una cierta relación, el único papel que juega la relación es como reductora de otras relaciones. Para una relación de este tipo no se necesita calcular la junta completa con ella (porque sus valores serán eliminados en seguida por una proyección); sólo necesita participar como reductora en programas de semijuntas. Las posibilidades de simplificación derivadas de estas observaciones no se consideran en ninguna de las referencias consultas, y por motivos de simplificación tampoco se consideran en el ORD.

4.6. Manejo de subexpresiones comunes

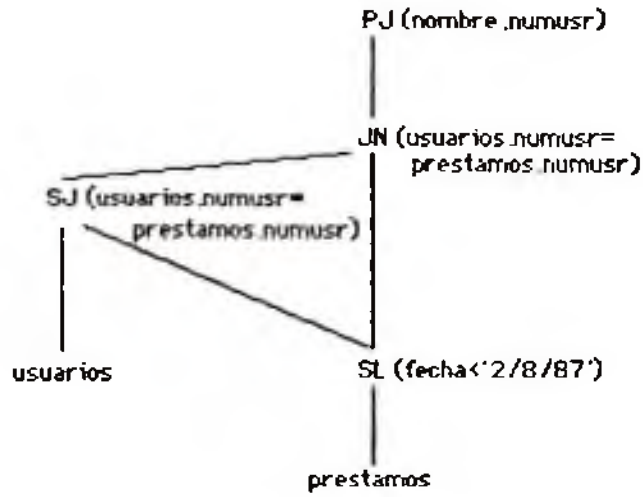
Como ya se mencionó (véase 2.2.2.), el manejo de subexpresiones comunes se vuelve necesario para hacer una estimación adecuada del costo de ejecución de una consulta, si se incluye la operación de semijunta. Algebraicamente, la introducción de una semijunta duplica por fuerza parte de la expresión, y puede llevar a una reducción en el costo de evaluación si la subexpresión duplicada se calcula una sola vez.

Para el reconocimiento de subexpresiones comunes se puede representar la expresión mediante una DAG (gráfica dirigida acíclica), como se hace en compiladores [3]. Una DAG es un árbol de sintaxis, usado también en compiladores, en el que dos subárboles iguales se representan una sola vez, dando lugar a una gráfica en la que no hay operaciones duplicadas.

Los vértices (internos) de la gráfica corresponden a la aplicación de una cierta operación, cuyo costo de ejecución se estima de acuerdo con las reglas de evaluación de atributos. El costo estimado de ejecución de una consulta se va a calcular como la suma de los costos asociados a cada vértice de la gráfica.

Un compilador también puede usar una DAG para generar el código que calcula una expresión, haciendo un recorrido de primero-profundidad sobre la gráfica para visitar exactamente una vez cada vértice. De la misma manera, este módulo del ORD se encarga de generar los planes de acceso en forma de programa. Esto se aprecia en el ejemplo siguiente.

> Ejemplo 4.3. Con referencia a la consulta del ejemplo 2.3., en la Fig. 4.5. (a) aparece la DAG correspondiente; nótese cómo la subexpresión duplicada aparece una sola vez.
> En la Fig. 4.5. (b) se muestra un plan de acceso en la forma de programa generado a partir de la DAG. El plan de acceso que se muestra es simplificado, la salida real del optimador para los ejemplos que se ha estado manejando aparece en el Apéndice 1.



(a) DAG correspondiente a una expresión de consulta.

```
e1 <- SL(fecha<'2/8/87') prestamos
e2 <- PJ(prestamos.numusr) e1
e3 <- ship(1) e2
e4 <- SJ(usuarios.numusr=prestamos.numusr) usuarios, e3
e5 <- ship(2) e4
e6 <- JN(usuarios.numusr=prestamos.numusr) e1,e5
r  <- PJ(titulo,numlibro) e6
```

(b) Plan de acceso en forma de programa.

Fig. 4.5. Representación mediante DAG y planes de acceso.

4.7. La máquina de búsqueda

La máquina de búsqueda que utiliza el ORD está definida de manera general por el OG como una modificación del algoritmo 3.1., y se define con base en algoritmo siguiente.

Algoritmo 4.4. Máquina del OG que busca una solución.
Descripción. Encuentra un plan de acceso eficiente para llevar a cabo una consulta, con base en las reglas de evaluación y cálculo dadas por el usuario.
Método.

1. Encuentra una expresión extendida inicial para la consulta del usuario; llámala expresión inicial.
2. Inicializa mejor a la expresión inicial. Inicializa el conjunto Abierto a que contenga justamente a mejor. Inicializa el conjunto Cerrado como vacío.
3. Si el conjunto Abierto está vacío o se satisface una condición de Terminación proporcionada por el usuario, termina. Usa el módulo de subexpresiones comunes para generar el plan de acceso correspondiente a la expresión mejor.
4. Excluye de Abierto la expresión x cuyo costo sea mínimo, de acuerdo con las reglas de evaluación del usuario y el módulo de subexpresiones comunes. Incluye x en el conjunto Cerrado.
5. Para cada expresión y a la que nos lleve una regla de transformación aplicada sobre x, aplica las reglas de mejora tantas veces como sea posible para obtener una expresión z; si z no está en Cerrado ni en Abierto y su costo estimado no sobrepasa al costo de la mejor expresión encontrada en un cierto factor de ascenso, inclúyela en Abierto, y si su costo es menor que el de mejor, remplaza mejor por z.
6. Continua con el paso 3.

La condición de terminación usada en el ORD puede corresponder a la búsqueda acotada en el tiempo, junto con un límite de memoria a usar. Otra alternativa es el uso de una búsqueda voraz, en la que el paso 5 del algoritmo anterior es remplazado por el siguiente:

5. Para cada expresión y a la que nos lleve una regla de transformación aplicada sobre x, aplica las reglas de mejora tantas veces como sea posible para obtener una expresión z. Para todas las expresiones z que se generen, elige la que tenga costo mínimo w. Si su costo es menor que el de mejor, remplaza mejor por w e incluye w en Abierto.

En los ejemplos se manejan factores de ascenso de 1.0 y 1.05, así como también búsqueda voraz.

4.8. Módulo de atención al usuario

En el sistema de prueba para el ORD es necesario construir un módulo de atención al usuario. Este módulo debe analizar una consulta formulada por el usuario y transformarla a su forma algebraica, además de desplegar en pantalla las expresiones transformadas que el optimador encuentra como solución.

En el módulo se usan variables cuyo valor son expresiones. El usuario formula las consultas con una notación semejante a la que se ha usado a lo largo del trabajo; los operandos son referencias a una variable o a una relación, y los esquemas son un parámetro del módulo. El nombre de una variable es de un carácter y para usarla una en una expresión se le precede por el carácter $.

Los comandos del lenguaje se componen de una variable seguida de un operador y, opcionalmente, una expresión. El operador = asigna a la variable la expresión, que se despliega también en pantalla; el operador ? despliega en pantalla la expresión que tiene asociada la variable, junto con su costo estimado; finalmente, el operador ! asigna a la variable el resultado de la optimación de la expresión, reporta en pantalla el factor de mejora que se logró, despliega la expresión optimada y escribe en memoria secundaria el plan de acceso correspondiente. A continuación se presenta un ejemplo.

> Ejemplo 4.4. En la Fig. 4.6. se muestra un ejemplo de diálogo con el usuario. Lo que el usuario escribe aparece subrrayado. Los planes de acceso generados aparecen en el Apéndice 1.

Este módulo se encarga también de la impresión de una expresión en pantalla; como se ve en el ejemplo anterior, esto se hace tratando de reflejar su estructura algebraica.

```
Optimación de consultas basada en reglas (CIEA-IPN, Agosto 1987).

> a! pj:l.titulo l.numlibro :sl:(l.autor='Aho'):l-libros

reducción de 340000.67:1
pj simple : l.titulo, l.numlibro
   sl indice : l.autor='Aho'
      l - libros
Costo: 3

> b= sl:(u.numusr=p.numusr):pd:u-usuarios,p-prestamos

sl simple : u.numusr=p.numusr
   pd ciclo 0
      u - usuarios
      p - prestamos

> c! pj:u.nombre u.numusr :sl:(p.fecha<'2/8/87'):%b

reduccion de 255.30:1
pj simple : u.nombre, u.numusr
   jn indice : u.numusr=p.numusr
      pj simple : p.numusr
         sl simple : p.fecha<'2/8/87'
            p - prestamos
      u - usuarios
Costo: 19846

> fin
```

Fig. 4.6. Un ejemplo de diálogo con el usuario.

4.9. Parámetros el optimador

De acuerdo con lo expuesto en este capítulo, el ORD maneja los siguientes parámetros:

- El sitio en que se requiere la respuesta a una consulta. Esta información se usa en la estimación del costo de una expresión y para formar alternativas en cuanto a programas de transmisión (véase 4.1.).
- El factor de ascenso para la máquina de búsqueda (véase 4.7.).
- El factor de optimación. Este factor determina la proporción de tiempo que se invierte en optimación, de acuerdo con el esquema de búsqueda acotada en el tiempo (véase 4.7.).
- Tiempo de acceso a memoria secundaria. Este parámetro sirve para estimar el tiempo que tardará una consulta (véase 4.1.).
- Octetos por página (véase 4.1).
- Factor de transmisión. Este parámetro determina en qué proporción es más cara la transmisión de datos que el acceso a memoria secundaria (véase 4.1.).
- Número de elemèntos que componen un dominio. Esto sirve para el cálculo de la selectividad de un campo (véase 4.5.2).
- Esquemas relacionales. Estos esquemas sirven para poder formar una expresión a partir de la consulta formulada por el usuario (véase 4.8.).

Los parámetros más importantes son, sin embargo, las reglas de evaluación, transformación y mejora, que hacen el ORD a partir del OG.

5. COMPARACION CON OTROS METODOS Y SISTEMAS

En este capítulo se comparan los resultados que obtiene el ORD con los resultados que obtienen otros métodos de optimación para bases de datos relacionales, tanto centralizadas como distribuidas. Después, se compara el OG con otras propuestas de optimadores basados en reglas.

5.1. Optimadores para el modelo relacional

Para bases de datos centralizadas, la comparación se hace con el método de Smith - Chang [33]; para la comparación con este método se emplea nuestra base de datos de la biblioteca, con una consulta un poco más elaborada que las que hemos usado en los ejemplos, tomada de [34]. Para bases de datos distribuidas, la comparación se hace con el método de Apers - Hevner - Yao [4,24] y con el del sistema SDD-1 [9]; en ambos casos, se usa el ejemplo que tiene la propia referencia. Aunque hay otros métodos para la optimación de consultas, los elegidos para estas comparaciones se consideran representativos.

A diferencia del ORD, los métodos anteriores usados para bases de datos distribuidas se aplican en consultas formadas exclusivamente por operaciones de junta, y se basan en una estrategia de dos etapas; primero se reducen las relaciones involucradas en la consulta, usando operadores unarios y programas de semijuntas, y luego se transmiten a un sitio donde se realizan las juntas. A la primera etapa se le llama de reducción; a la segunda, de ensamble.

Cada referencia usa una representación propia que permite visualizar la operación del método. Para efecto de comparar las estrategias generadas, las representaremos mediante gráficas de flujo [9]. En una gráfica de flujo los vértices representan relaciones y se colocan en la columna que corresponde al sitio donde se obtienen. Los arcos pueden conectar dos vértices de la misma columna, para representar una operación local, o vértices que estén en columnas distintas, en cuyo caso representan una transmisión, como se verá en los ejemplos. Los planes de acceso y expresiones que encuentra el ORD aparecen en el apéndice 1.

5.1.1. El método de Smith - Chang

Este método se propone en [33], y es una de las primeras referencias sobre optimación de consultas. La idea principal del trabajo es la realización de las operaciones unarias tan pronto como sea posible, con el propósito de reducir el tamaño de los resultados intermedios. El sistema considera, además, un procedimiento de selección de método basado en los campos sobre los cuales puede quedar clasificada una relación intermedia.

En la referencia se usa una representación algebraica semejante a la usada en el ORD, y también se tiene un repertorio de métodos de ejecución para cada operación del álgebra relacional. Se propone el uso de reglas, que son usadas para determinar los métodos de ejecución. Este método es para bases

de datos centralizadas, por lo que no considera costos de transmisión. El procedimiento de optimación propuesto se describe a continuación.

Algoritmo 5.1. Metodo de optimación de Smith - Chang.
Descripción. Dada una consulta en álgebra relacional, encuentra una expresión equivalente pero más eficiente y asocia a cada operación un método de ejecución.
Método.

1. Usa la propiedad de distributividad de la selección respecto de los operadores binarios, para llevar esta operación tan cerca como sea posible de las relaciones que toman parte en la consulta.
2. Usando la propiedad distributiva de la proyección respecto de los operadores binarios, lleva esta operación hacia las relaciones de la consulta. Si alguna de éstas se aplica sobre una relación con índices, elimínala.
3. Etiqueta cada operador con el conjunto de campos sobre los cuales puede quedar clasificada la relación resultante. Como para determinar esto necesitamos saber los campos sobre los que pueden estar clasificadas las relaciones argumento, este es el "paso hacia arriba".
4. Dependiendo de la clasificación que se requiera para la relación final, determina qué clasificación usar en cada subexpresión; de acuerdo con esto, elige el método más eficiente que obtenga la relación clasificada como se desea. Este es el paso "hacia abajo".

Los pasos 3 y 4 están codificados en la referencia como una serie de reglas. Las reglas "hacia arriba" determinan el conjunto de campos que se calcula en el paso 3, basadas en la operación de que se trate y los campos de clasificación de los argumentos. Las reglas "hacia abajo" determinan el método a usar de manera semejante.

A continuación vemos un ejemplo de optimación siguiendo el método recién descrito, donde se le compara con el resultado obtenido con el ORD.

Ejemplo 5.1. Con referencia a nuestra base de datos del ejemplo 1.2., queremos saber los títulos de los libros que han sido prestado con fecha anterior al 2/8/87. Suponemos que el contenido de la base de datos está resumido en los perfiles de la Fig. 1.6. La expresión algebraica de la consulta aparece en la Fig. 5.1.
En la Fig. 5.2. (a) aparece la expresión solución que se encuentra por el método de Smith - Chang, mientras que en la Fig. 5.2. (b) se muestra la expresión obtenida por el ORD con el método voraz. La expresión encontrada es la misma, y su costo, en número estimado de páginas de memoria secundaria, a accesar se aprecia en la Fig. 5.2. (c).

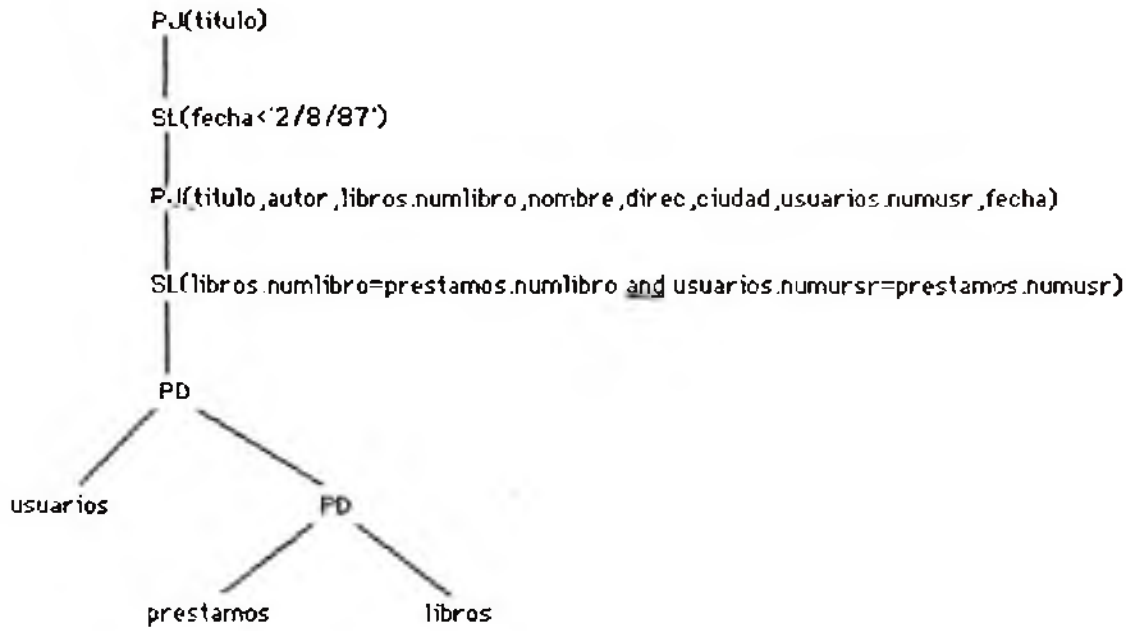
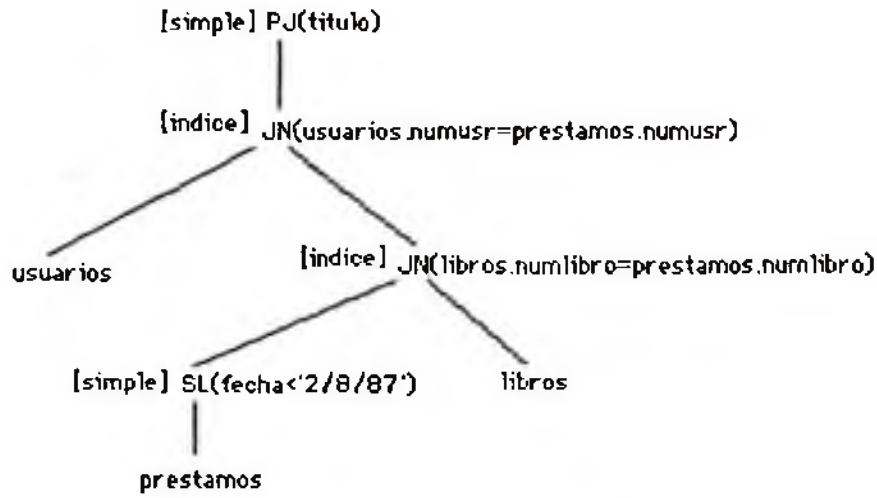
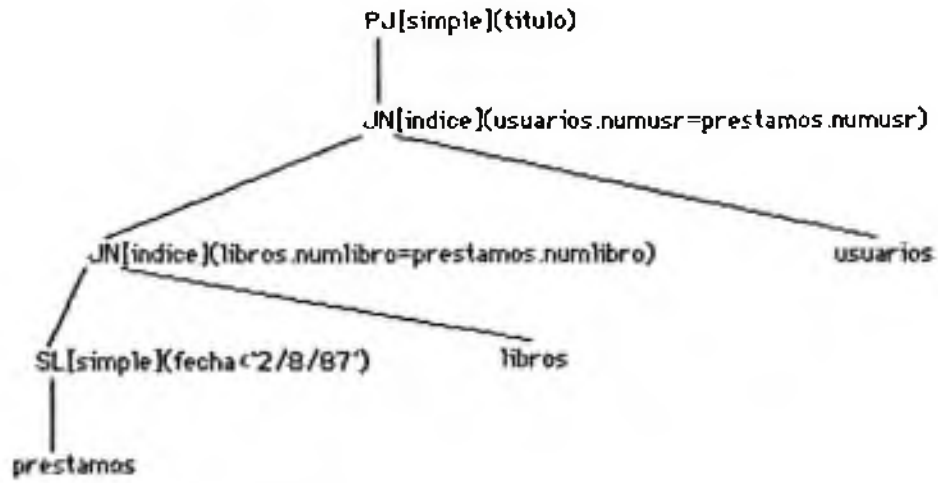


Fig. 5.1. Expresión algebraica de una consulta.



(a) Expresión solución por el método de Smith - Chang.



(b) Expresión solución encontrada por el ORD.

	Inicial	Smith - Chang	ORD
Páginas de memoria secundaria a accesar	1.94E13	44133	44133

(c) Costo de la expresión inicial y de las expresiones solución.

Fig. 5.2. Expresiones solución para la consulta de la Fig. 5.1.

5.1.2. El método de Apers - Hevner - Yao

Este es un método de optimación de consultas para bases de datos distribuidas, y se propone en [4,24]. El método se basa en la noción de consulta simple. Una consulta simple se realiza sobre un conjunto de relaciones que tienen un solo atributo. La manera óptima de resolver una consulta simple es de acuerdo a los tamaños de las relaciones involucradas: la relación más pequeña se envía a la segunda más pequeña para realizar la junta correspondiente, que se manda a la tercera más pequeña y así sucesivamente hasta la última. El resultado de la junta queda en el sitio de la relación más grande de todas.

La estrategia de ejecución para consultas simples puede usarse para resolver consultas en las que las relaciones tienen más de un atributo, como las que se esperaría usar en una base de datos real; esto se hace utilizando la solución de una consulta simple como un programa de semijuntas para reducir relaciones. Aunque la solución de una consulta simple es óptima usando la estrategia anterior, esta optimidad se pierde al integrar las soluciones para resolver una consulta compleja. Hay varias formas de integrar consultas simples en programas de reducción, algunas de estas formas persiguen minimizar el costo de la consulta, en términos de datos transmitidos, mientras que otras tratan de minimizar el tiempo de respuesta, tratando de explotar al máximo el paralelismo en la red. Vamos a considerar el algoritmo COLLECT [4], que tiene por objeto la minimización de los costos de transmisión.

Algoritmo 5.2. Formación de reducciones COLLECT (AHY).
Descripción. Dada una consulta, se determinan programas de semijunta reductores para cada relación que participa en ella. Cuando tenemos que dos o más relaciones están siendo usadas en una junta y se comparan campos con el mismo dominio para cada una de ellas, podemos plantear una consulta simple considerando que cada relación tiene sólo el campo usado en la comparación. Los programas de semijuntas se basan en la solución de estas consultas simples. Las relaciones, una vez reducidas, se envían al sitio donde se requiere la respuesta y allí se realizan las operaciones de junta.
Método.

1. Para cada dominio cuyos campos se usen en una condición de junta, construye una consulta simple cuyas relaciones están formadas por los campos que pertenecen al dominio y son comparados en las condiciones, junto con las condiciones mismas.
2. Resuelve las consultas simples obtenidas en el paso 1.
3. Para cada relación, forma un programa de reducción con las soluciones de las consultas simples en que se encuentren involucrados campos de la relación. La transmisión de un campo a una cierta relación puede formar parte de varios programas de reducción; al calcular el costo de las reducciones, estas transferencias se cuentan sólo una vez.
4. Considera el costo de la reducción si se elimina de

todos los programas de semijunta la transmisión de un cierto campo a un sitio. Siempre que convenga, climina esta transmisión.

Las relaciones reducidas mediante el algoritmo anterior son luego transmitidas al sitio de respuesta, donde se ejecutan las juntas de la consulta. En el ejemplo siguiente se resulve una consulta usando este algoritmo y se compara el resultado con el obtenido por el ORD.

Ejemplo 5.2. En la Fig. 5.3. (a) aparecen las relaciones de una base de datos hipotética, junto con algunos datos de sus campos que sirven para el cálculo de la selectividad. En la Fig. 5.3. (b) se muestra la consulta que nos interesa, y en (c) la gráfica de consulta correspondiente. Suponemos que el resultado de la consulta se requiere en un sitio distinto de donde están almacenadas las relaciones.
En la Fig. 5.4. (a) se muestra la gráfica de flujo correspondiente a la estrategia determinada por el método de Apers - Hevner - Yao. Las columnas están etiquetadas con el nombre de la relación que se encuentra en cada sitio. Cada punto que aparece en una columna corresponde a una relación intermedia; junto a la relación aparece la operación usada para obtenerla. Los arcos diagonales representan una transmisión de datos entre sitios; están etiquetados por el campo que se transmite y el costo de transmisión.
En la Fig. 5.4. (b) aparece la gráfica de flujo de la estrategia encontrada por el ORD con un parámetro de ascenso de 1.05. Por último, en la Fig. 5.4. (c) se muestran los costos de ejecución de cada una de las estrategias, medidos en número de octetos a transmitir. La estrategia inicial corresponde a la transmisión de las tres relaciones completas al sitio de usuario y allí ejecutar las juntas.

Relación r1

Cardinalidad : 2000
Sitio : 1

Campo	Ancho	Valores distintos
c1	1	2000
c2	2	1000
c3	2	150

Relación r2

Cardinalidad : 5000
Sitio : 2

Campo	Ancho	Valores distintos
c1	1	2000
c2	2	1200
c3	2	450

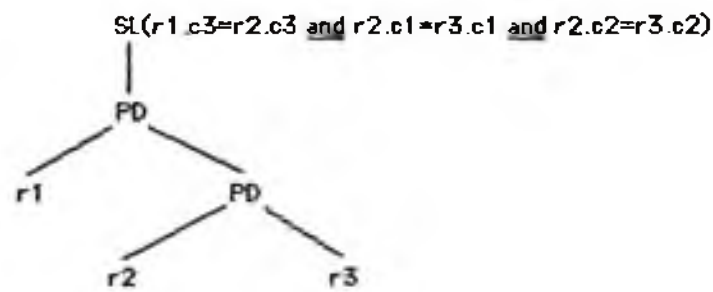
Relación r3

Cardinalidad : 2500
Sitio : 3

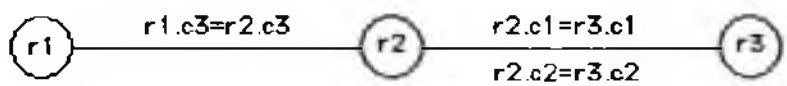
Campo	Ancho	Valores distintos
c1	1	1500
c2	2	900
c3	2	150

Valores distintos en el dominio de c1 : 5000
Valores distintos en el dominio de c2 : 1500
Valores distintos en el dominio de c3 : 1500

(a) Perfiles de las relaciones de una base de datos.

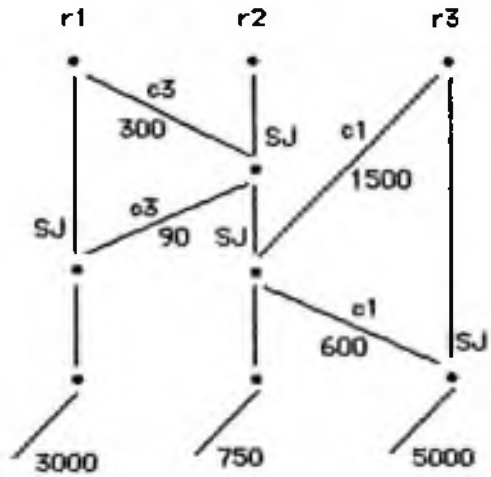


(b) Expresión relacional de una consulta.

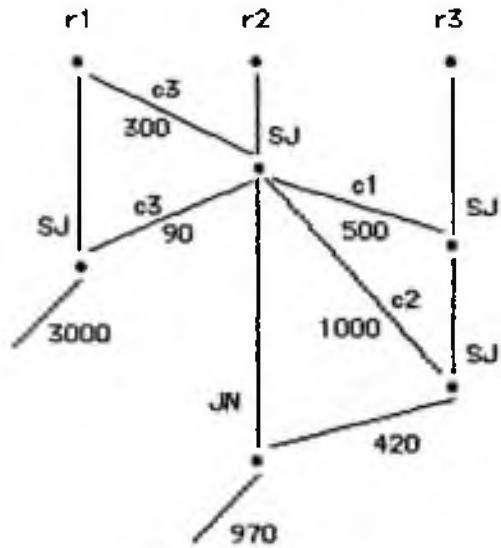


(c) Gráfica de consulta correspondiente.

Fig. 5.3. Perfil de una base de datos y una consulta sobre ésta.



(a) Estrategia solución por el método de Apers - Hevner - Yao.



(b) Estrategia solución encontrada por el ORD.

	Inicial	Apers - Hevner - Yao	ORD
Número de octetos a transmitir	47500	11240	6280

(c) Costo de la estrategia inicial y de las estrategias solución.

Fig. 5.4. Estrategias solución para la consulta de la Fig. 5.3.

5.1.3. El sistema SDD-1

SDD-1 es un sistema para bases de datos distribuidas; el procedimiento de optimación que sigue se presenta en [9]. El método determina de manera voraz las semijuntas que es conveniente realizar, en términos del volumen de datos que se transmiten para realizar la semijunta y la reducción que se tiene como efecto de ésta; así se forma un programa de reducción inicial. Con base en el programa inicial se elige un sitio para llevar a cabo la etapa de ensamble. De acuerdo con este sitio se pueden luego eliminar algunas semijuntas o cambiar el orden en que se realizan. El algoritmo se muestra a continuación.

Algoritmo 5.3. Formación de reducciones en SDD-1.
Descripción. Con base en una gráfica de consulta, se determina de manera voraz un programa de reducción para las relaciones.
Método.

1. Para cada arista de la gráfica de consulta, calcula el beneficio de la reducción de las relaciones que ésta une, como el volumen de datos en que se reduce la relación menos el volumen de datos que se requiere transmitir para ejecutar la semijunta. Elige la reducción con mayor beneficio e incluye la semijunta en el programa reductor. Repite este paso (vorazmente) hasta que no queden reducciones con beneficio.
2. Elige como sitio de ensamble el de la relación cuyo volumen de datos sea mayor después de las reducciones.
3. Elimina las semijuntas que sólo reducen a la relación que ya está en el sitio de ensamble, y reordena las semijuntas de modo que se realicen primero las que requieren un menor costo de transmisión.

El paso 1 del algoritmo encuentra un programa de reducción inicial, mientras que el paso 3 corresponde a lo que la referencia llama una etapa de postoptimación, para compensar la voracidad del algoritmo en la selección del programa de reducción inicial. A continuación se presenta un ejemplo.

Ejemplo 5.3. En la Fig. 5.5. (a) aparecen las relaciones de una base de datos hipotética, junto con algunos datos de sus campos que sirven para el cálculo de la selectividad. En la Fig. 5.5. (b) se muestra la consulta que nos interesa, y en (c) la gráfica de consulta correspondiente.
En la Fig. 5.6. (a) aparece la gráfica de flujo para la estrategia encontrada por el método de SDD-1, el resultado de la consulta queda en el sitio de y. En la Fig. 5.6. (b) aparece la gráfica de flujo para estrategia encontrada por el ORD con un parámetro de ascenso de 1.05. Nótese que, aunque las estrategias son un poco distintas, sus costos de transmisión son iguales.
En la Fig. 5.6. (c) aparecen los costos de cada una de las estrategias, medidos en el número de octetos que se necesita transmitir. La estrategia inicial es transmitir las relaciones s y p completas al sitio de la relación y.

Relación s

Cardinalidad : 10000
Sitio : 1

Campo	Ancho	Valores distintos
s	1	10000
nombre	1	10000
localidad	1	50

Relación y

Cardinalidad : 100000
Sitio : 2

Campo	Ancho	Valores distintos
s	1	1000
p	1	1000

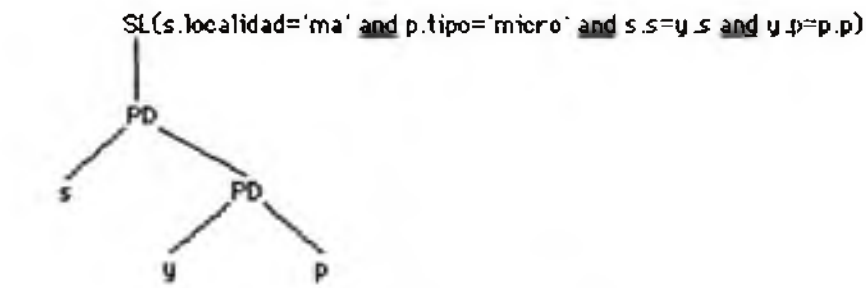
Relación p

Cardinalidad : 10000
Sitio : 3

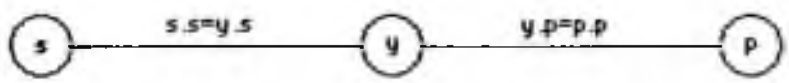
Campo	Ancho	Valores distintos
p	1	10000
nombre	1	10000
tipo	1	5

Valores distintos en el dominio de s : 10000
Valores distintos en el dominio de p : 10000

(a) Perfiles de las relaciones de una base de datos.

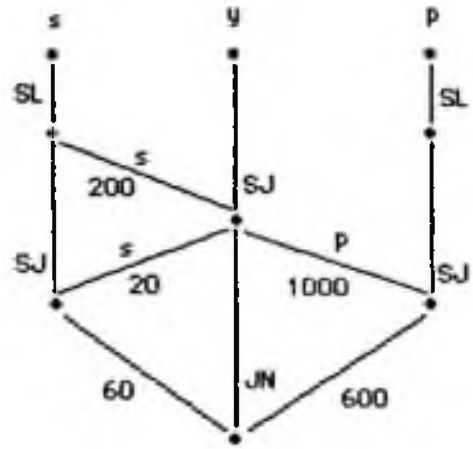


(b) Expresión relacional de una consulta.

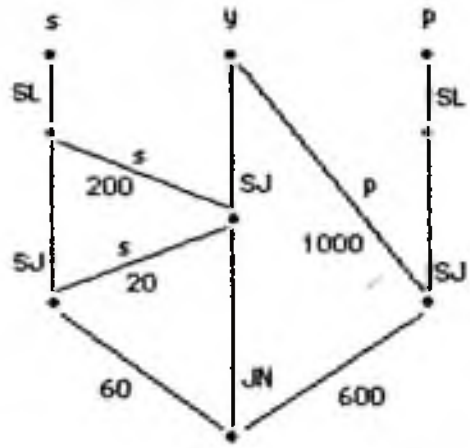


(c) Gráfica de consulta correspondiente.

Fig. 5.5. Perfil de una base de datos y una consulta sobre ésta.



(a) Estrategia solución por el método del sistema SDD-1.



(b) Estrategia solución encontrada por el ORD.

	Inicial	SDD-1	ORD
Número de octetos a transmitir	6200	1880	1880

(c) Costo de la estrategia inicial y de las estrategias solución.

Fig. 5.6. Estrategias solución para la consulta de la Fig. 5.5.

5.2. Optimadores de consultas basados en reglas

El uso de un enfoque transformacional para la optimación de consultas se recomienda principalmente en [12,13,37]. La construcción de optimadores basados en reglas se sugiere a raíz de la aparición de bases de datos orientadas a objetos [11,19,23,29], y es relativamente reciente. Una base de datos orientada a objetos es extensible en cuanto a los operandos y operadores que maneja, y por tanto necesita un optimador que sea extensible también. Hay varios trabajos al respecto, pero se encuentran a nivel experimental y no se tienen aún conclusiones definitivas. El estado de desarrollo de estos trabajos se observa en la sección de optimación basada en reglas de [1].

Aparentemente el sistema más completo es el que se desarrolla como parte del proyecto EXODUS [11,23]. En este sistema se sigue un enfoque de generación del optimador. A partir de un archivo de descripción se genera un optimador escrito en lenguaje C; en este archivo se colocan las reglas de transformación y métodos de ejecución para cada operador en un formato especial, y se pueden incluir, además, rutinas en C. No hay una clasificación de reglas de mejora y transformación; se tienen reglas de transformación con un valor asociado que refleja el factor de reducción esperado después de su aplicación. Lo que equivale a nuestra reglas de evaluación se programa directamente en C en el archivo de descripción. La selección de método se hace individualmente de manera voraz para cada operador, y no se tienen transformaciones de cambio de método. Esto se debe a que no se manejan transformaciones sobre expresiones extendidas, sino sobre expresiones que después se "extienden" vorazmente para calcular su costo.

En [23] se presenta un optimador para un subconjunto del modelo relacional basado en las convenciones del sistema EXODUS. Sólo se incluyen las operaciones SL y JN, y las reglas de transformación pertinentes a estas operaciones. Las relaciones se suponen almacenadas en el mismo sitio. Aparentemente, el enfoque de generación que se sigue produce optimadores muy eficientes. En esta referencia se sugiere la incorporación de un módulo de manejo de predicados como parte del optimador general. Nuestro propio módulo, descrito en 4.5., puede ser un paso en esta dirección.

En [25] se presenta la aplicación de un método distinto de exploración del espacio aplicado a la optimación de consultas. El método se denomina de solidificación (annealing), porque semeja la estabilización gradual (una función del calor y el tiempo) que ocurre durante la solidificación de ciertos materiales cuando se les dejan enfriar después de ser calentados.

En este tipo de búsqueda no se tiene la noción de mejor, como se maneja en el algoritmo 1.2., sino la noción de actual. Se maneja un parámetro semejante al factor de ascenso, que en este método es una función del tiempo. actual se mueve entre estados vecinos dependiendo de su costo con una cierta probabilidad, que depende del factor de ascenso; cuando se alcanza cierta "estabilidad", definida según el problema, se

reduce el factor de ascenso que, al alcanzarse el punto de solidificación, no permite ya movimiento alguno a actual. El estado en que queda Actual es la solución.

La aplicación es en la optimación de una clase restringida de consultas recursivas.

En [21] se presenta un enfoque de transformación basada en reglas donde no se usa estimación de costos para guiar la búsqueda, sino que sólo se aplican reglas que garantizan mejoras. Tiene referencia a otro sistema, basado en reglas también, que forma un programa de alto nivel a partir de una consulta dada en forma algebraica [22]. Estos sistema se apoyan en trabajos relacionados con transformación de programas; en ese contexto no hay, en general, un modelo de costo que nos diga si un programa es mejor que otro, y mucho menos en qué proporción. Algunos sistemas de transformación de programas incluso operan guiados por el usuario [18].

En [28] se propone un sistema basado en reglas para hacer uso de restricciones de integridad en lo que la propia referencia llama optimación semántica. Se identifican varios tipos de transformación que pueden realizarse sobre esta base. La optimación de la consulta se realiza en varias etapas; en la primera de ellas se aplican las transformaciones basadas en información semántica sin hacer evaluación de costos; en la última etapa se eligen los métodos de ejecución, ahora sí con base en el costo estimado. De este modo puede establecerse una relación con nuestra reglas de mejora.

Usando restricciones de integridad como las que se manejan en esta referencia, se puede hacer una simplificación adicional a la consulta de ejemplo 5.1.; se puede eliminar la junta con la relación usuarios. Para llegar a este resultado, no es suficiente considerar las propiedades algebraicas de los operadores relacionales, sino más bien observar la restricción de integridad que establece que para todo numusr en préstamos, debe existir un numusr en usuarios.

Una característica importante de los trabajos de las dos últimas referencias es que no tienen una forma única de representación para las consultas; esto es a diferencia del ORD, que maneja expresiones extendidas exclusivamente. El propio planteo de la optimación, por etapas, admite de manera natural cambios en la representación de la consulta. También es importante notar que en ocasiones no se necesita estimar el costo, y se puede usar una representación menos detallada.

Conclusiones

Hasta donde sabemos, el optimador para base de datos relacionales ORD es la aplicación más completa que se ha hecho hasta el momento de la filosofía de optimadores basados en reglas que manejan expresiones algebráicas. La experiencia con este trabajo ha dado lugar a varias observaciones.

1. Varios métodos de optimación pueden encajar en un marco transformacional. Considérense, por ejemplo, los métodos de Apers - Hevner - Yao y del sistema SDD-1; se reconoce el esquema de partir de una solución inicial y aplicar una serie de transformaciones (de manera voraz) para llegar a la solución.
2. La representación algebráica no es la más adecuada en todos los casos. Aunque es un enfoque muy general, puede haber situaciones en que no sea fácil de usar. Considérese, por ejemplo, la situación de la optimación en base de datos distribuidas; aunque la representación de programas de semijuntas en forma algebráica es factible, con una representación como la de gráficas de flujo las estrategias son mucho más claras. Más aun, se podrían codificar reglas de transformación sobre gráficas de flujo que fueran más naturales al problema en cuestión y cuya aplicación implicara menos búsqueda de la que se emplea con una representación algebráica. Considérense también la detección y solución de consultas cíclicas; el problema no encaja naturalmente en una representación algebráica.

A partir de estas observaciones se realizan las siguientes propuestas respecto de la filosofía y aplicación futura de los optimadores basados en reglas.

1. Puede ser conveniente el uso de varias formas de representación a lo largo de la solución del problema. Tenemos representaciones adecuadas para resolver algunos aspectos bien definidos, pero que no sirven para resolver apropiadamente otros. Considérese, por ejemplo, el tratar de usar gráficas de flujo para resolver el problema de la optimación de principio a fin; la selección de método y manejo de operadores unarios no encaja de manera natural.
2. Puede ser conveniente el uso de patrones de transformación. La solución de un problema por etapas (primero resuelve este aspecto, luego aquel, etc.) no lleva necesariemente a la solución del problema. Podrían especificarse patrones de aplicación de reglas en base a autómatas finitos, por ejemplo. Cada consulta que se genera durante la búsqueda puede tener asociado un estado del autómata, que determina la representación que ha de tener y también las reglas que se le pueden aplicar. Con un esquema como éste se puede seguir teniendo la flexibilidad de un sistema basado en reglas, además de hacer uso de varias formas de representación.

Apéndice 1. Salida del optimador para los ejemplos usados

En este apéndice se presentan los planes de acceso que el ORD encuentra para los ejemplos que se manejaron en el trabajo, de la manera como los escribe en su bitácora. Dada una expresión extendida, se estima su costo como el número de páginas que es necesario accesar; el costo de una transmisión se mide también en accesos a memoria secundaria, de acuerdo con el factor de transmisión. Considerando un tiempo de acceso de 3E-3 segundos por página se estiman los tiempos de ejecución que se reportan.

Cada línea del plan de acceso corresponde al cálculo de una relación intermedia, que queda asociada a la variable que se coloca del lado izquierdo. En la primera columna se indica en qué sitio está la relación calculada.

1. Salida para la consulta del ejemplo 1.4.

```
pj simple : b.title,b.lc_no
   sl simple : b.author='Aho'
      b - books :

metodo ASCENSO-DE-COLINA(1.05)
1 expresiones generadas.
3 expresiones expandidas.
298.83 segundos de ejecucion exp inicial.
0.01 segundos de ejecucion de solucion.
reduccion de 33203.79 : 1.

pj simple : b.title,b.lc_no
   sl indice : b.author='Aho'
      b - books :

 1:    e0 <-- b - books
 1:    e1 <-- sl indice b.author='Aho' : e0
 1:    i0 <-- pj simple b.title,b.lc_no : e1
 1:    e2 <-- sort("b.title") < store < i0
 1:   res <-- e2
```

2. Salida para la consulta del ejemplo 1.5.

```
pj simple : w.name,w.card_no
   sl simple : w.card_no=l.card_no AND l.date<'2/8/87
      pd loop 0
         w - borrowers :
         l - loans :

metodo ASCENSO-DE-COLINA(1.05)
30 expresiones generadas.
10 expresiones expandidas.
15200.05 segundos de ejecucion exp inicial.
59.54 segundos de ejecucion de solucion.
reduccion de 255.30 : 1.

pj simple : w.name,w.card_no
   jn indice 0 : w.card_no=l.card_no
      pj simple : l.card_no
         sl simple : l.date<'2/8/87'
            l - loans :
      w - borrowers :

 1:    e0 <-- l - loans
 1:    i0 <-- scan("") < e0
 1:    e1 <-- sl simple l.date<'2/8/87' : i0
 1:    i1 <-- pj simple l.card_no : e1
 1:    e2 <-- sort("l.card_no") < store < i1
 1:    e3 <-- w - borrowers
 1:    i2 <-- scan("") < e2
 1:    e4 <-- jn indice w.card_no=l.card_no : i2 e3
 1:    i3 <-- pj simple w.name,w.card_no : e4
 1:    e5 <-- sort("w.name") < store < i3
 1:   res <-- e5
```

3. Salida para la consulta del ejemplo 2.2.

```
pj simple : b.title,b.lc_no
   sl simple : b.author='Aho' AND b.library='4
      un simple 0 : () (b3->b)
         un simple 0 : (b1->b) (b2->b)
            b1 - books1 : b1.library<'4'
            b2 - books2 : b2.library='4'
         b3 - books3 : b3.library='5'

metodo ASCENSO-DE-COLINA(1.05)
1 expresiones generadas.
3 expresiones expandidas.
404162.65 segundos de ejecucion exp inicial.
3.01 segundos de ejecucion de solucion.
reduccion de 134184.15 : 1.

pj simple : b.title,b.lc_no
   sl indice : b.author='Aho'
      b - books2 : b.library='4'

 2:    e0 <-- b - books2
 2:    e1 <-- sl indice b.author='Aho' : e0
 2:    i0 <-- pj simple b.title,b.lc_no : e1
 2:    e2 <-- sort("b.title") < store < i0
 1:  res <-- ship(1) < scan("") < e2
```

4. Salida para la consulta del ejemplo 2.3.

```
pj simple : w.name,w.card_no
   sl simple : w.card_no=l.card_no AND l.date<'1/2/87'
      pd loop 0
         w - borrowers :
         l - loans :

metodo ASCENSO-DE-COLINA(1.05)
56 expresiones generadas.
12 expresiones expandidas.
25992.83 segundos de ejecucion exp inicial.
2282.87 segundos de ejecucion de solucion.
reduccion de 11.39 : 1.

pj simple : w.name,w.card_no
   jn merge 2 : w.card_no=l.card_no
      pj simple : l.card_no
         sl simple : l.date<'1/2/87'
            l - loans :
      pj simple : w.name,w.card_no
         sj merge : w.card_no=l.card_no
            w - borrowers :
            pj simple : l.card_no
               sl simple : l.date<'1/2/87'
                  l - loans :

 2:    e0 <-- l - loans
 2:    i0 <-- scan("") < e0
 2:    e1 <-- sl simple l.date<'1/2/87' : i0
 2:    i1 <-- pj simple l.card_no : e1
 2:    e2 <-- sort("l.card_no") < store < i1
 1:    e3 <-- w - borrowers
 1:    i2 <-- scan("w.card_no") < e3
 1:    i3 <-- scan("") < sort("l.card_no") < ship(1) < scan("") < e2
 1:    e4 <-- sj merge w.card_no=l.card_no : i2 i3
 1:    i4 <-- pj simple w.name,w.card_no : e4
 1:    e5 <-- sort("w.name") < store < i4
 2:    i5 <-- scan("") < sort("l.card_no") < e2
 2:    i6 <-- scan("") < sort("w.card_no") < ship(2) < scan("") < e5
 2:    e6 <-- jn merge w.card_no=l.card_no : i5 i6
 2:    i7 <-- pj simple w.name,w.card_no : e6
 2:    e7 <-- sort("w.name") < store < i7
 2:   res <-- e7
```

5. Salida para la consulta del ejemplo 5.1.

```
pj simple : b.title
   sl simple : l.date<'02/08/82'
      pj simple : b.title,b.author,b.lc_no,w.name,w.addr,w.city,
                  w.card_no,l.date
         sl simple : b.lc_no=l.lc_no AND w.card_no=l.card_no
            pd loop 0
               pd loop 0
                  l - loans :
                  b - books :
               w - borrowers :

metodo ASCENSO-DE-COLINA(1.05)
148 expresiones generadas.
30 expresiones expandidas.
58122139441.85 segundos de ejecucion exp inicial.
132.40 segundos de ejecucion de solucion.
reduccion de 438994625.23 : 1.

pj simple : b.title
   jn indice 0 : w.card_no=l.card_no
      jn indice 0 : b.lc_no=l.lc_no
         sl simple : l.date<'02/08/82'
            l - loans :
         b - books :
      w - borrowers :

 1:   e0 <-- l - loans
 1:   i0 <-- scan("") < e0
 1:   e1 <-- sl simple l.date<'010182' : i0
 1:   e2 <-- b - books
 1:   e3 <-- jn indice b.lc_no=l.lc_no : e1 e2
 1:   e4 <-- w - borrowers
 1:   e5 <-- jn indice w.card_no=l.card_no : e3 e4
 1:   i1 <-- pj simple b.title : e5
 1:   e6 <-- sort("b.title") < store < i1
 1:  res <-- e6
```

6. Salida para la consulta del ejemplo 5.2.

```
sl simple : r1.c3=r2.c3 AND r2.c1=r3.c1 AND r2.c2=r3.c2
   pd loop 0
      pd loop 0
         r2 - r2 :
         r1 - r1 :
      r3 - r3 :

metodo ASCENSO-DE-COLINA(1.05)
153 expresiones generadas.
20 expresiones expandidas.
7302562.72 segundos de ejecucion exp inicial.
393.67 segundos de ejecucion de solucion.
reduccion de 18550.10 : 1.

jn indice 5 : r1.c3=r2.c3
   jn merge 0 : r2.c1=r3.c1 AND r2.c2=r3.c2
      sj merge : r2.c2=r3.c2
         sj merge : r2.c1=r3.c1
            r3 - r3 :
            pj simple : r2.c1
               sj loop : r1.c3=r2.c3
                  r2 - r2 :
                  pj simple : r1.c3
                     r1 - r1 :
         pj simple : r2.c2
            sj loop : r1.c3=r2.c3
               r2 - r2 :
               pj simple : r1.c3
                  r1 - r1 :
      sj loop : r1.c3=r2.c3
         r2 - r2 :
         pj simple : r1.c3
            r1 - r1 :
   sj loop : r1.c3=r2.c3
      r1 - r1 :
      pj simple : r2.c3
         sj loop : r1.c3=r2.c3
            r2 - r2 :
            pj simple : r1.c3
               r1 - r1 :

3:    e0 <-- r3 - r3
2:    e1 <-- r2 - r2
1:    e2 <-- r1 - r1
1:    i0 <-- scan("") < e2
1:    i1 <-- pj simple r1.c3 : i0
1:    e3 <-- sort("r1.c3") < store < i1
2:    i2 <-- scan("") < ship(2) < scan("") < e3
2:    e4 <-- sj loop r1.c3=r2.c3 : e1 i2
2:    i3 <-- pj simple r2.c1 : e4
2:    e5 <-- sort("r2.c1") < store < i3
3:    i4 <-- scan("") < sort("r3.c1") < e0
3:    i5 <-- scan("") < sort("r2.c1") < ship(3) < scan("") < e5
```

```
3:   e6 <-- sj merge r2.c1=r3.c1 : i4 i5
2:   i6 <-- pj simple r2.c2 : e4
2:   e7 <-- sort("r2.c2") < store < i6
3:   i7 <-- scan("") < sort("r3.c2") < store < e6
3:   i8 <-- scan("") < sort("r2.c2") < ship(3) < scan("") < e7
3:   e8 <-- sj merge r2.c2=r3.c2 : i7 i8
2:   i9 <-- scan("") < sort("r3.c1") < ship(2) < e8
2:  i10 <-- scan("") < sort("r2.c1") < store < e4
2:   e9 <-- jn merge r2.c1=r3.c1 AND r2.c2=r3.c2 : i9 i10
2:  i11 <-- pj simple r2.c3 : e4
2:  e10 <-- sort("r2.c3") < store < i11
1:  i12 <-- scan("") < ship(1) < scan("") < e10
1:  e11 <-- sj loop r1.c3=r2.c3 : e2 i12
5:  i13 <-- scan("") < ship(5) < e9
5:  i14 <-- index("r1.c3") < ship(5) < e11
5:  e12 <-- jn indice r1.c3=r2.c3 : i13 i14
5:  res <-- e12
```

7. Salida para la consulta del ejemplo 5.3.

```
sl simple : s.location='ma' AND p.type='micro' AND s.s=y.s AND y.p=p.p
   pd loop 0
      pd loop 0
         s - s :
         y - y :
      p - p :

metodo ASCENSO-DE-COLINA(1.05)
316 expresiones generadas.
38 expresiones expandidas.
464950225.27 segundos de ejecucion exp inicial.
184.09 segundos de ejecucion de solucion.
reduccion de 2525709.70 : 1.

jn loop 2 : s.s=y.s
   jn merge 2 : y.p=p.p
      sj loop : s.s=y.s
         y - y :
         pj simple : s.s
            sl simple : s.location='ma'
               s - s :
      sj merge : y.p=p.p
         sl simple : p.type='micro'
            p - p :
         pj simple : y.p
            y - y :
   sj loop : s.s=y.s
      sl simple : s.location='ma'
         s - s :
      pj simple : y.s
         sj loop : s.s=y.s
            y - y :
            pj simple : s.s
               sl simple : s.location='ma'
                  s - s :

 2:   e0 <-- y - y
 1:   e1 <-- s - s
 1:   i0 <-- scan("") < e1
 1:   e2 <-- sl simple s.location='ma' : i0
 1:   i1 <-- pj simple s.s : e2
 1:   e3 <-- sort("s.s") < store < i1
 2:   i2 <-- scan("") < ship(2) < scan("") < e3
 2:   e4 <-- sj loop s=y.s: e0 i2
 3:   e5 <-- p - p
 3:   i3 <-- scan("") < e5
 3:   e6 <-- sl simple p.type='micro' : i3
 2:   i4 <-- scan("") < e0
 2:   i5 <-- pj simple y.p : i4
 2:   e7 <-- sort("y.p") < store < i5
 3:   i6 <-- scan("") < sort("p.p") < store < e6
 3:   i7 <-- scan("") < sort("y.p") < ship(3) < scan("") < e7
 3:   e8 <-- sj merge y.p=p.p : i6 i7
```

```
2:   i8 <-- scan("") < sort("y.p") < store < e4
2:   i9 <-- scan("") < sort("p.p") < ship(2) < e8
2:   e9 <-- jn merge y.p=p.p : i8 i9
2:  i10 <-- pj simple y-s : e4
2:  e10 <-- sort("y.s") < store < i10
1:  i11 <-- store < e2
1:  i12 <-- scan("") < ship(1) < scan("") < e10
1:  e11 <-- sj loop s.s=y.s : i11 i12
2:  i13 <-- ship(2) < e11
2:  e12 <-- jn loop s.s=y.s : e9 i13
2:  res <-- e12
```

Apéndice 2. Listado del optimador

El optimador está programado en Turbo-Prolog. Es un "proyecto" con un archivo de definición dominios.pro. y se compone de los siguientes modulos:

- parametros.pro
- utilerias.pro
- parser.pro
- print.pro
- analisis.pro
- eval.pro
- subexp.pro
- reglas.pro
- maquina.pro / maquina2.pro
- dialogo.pro

Los archivos se listan a continuación.

```
/* dominios.pro */

/*
================================================================
*/
/* Declaraciones globales
*/


/*
================================================================
*/
/* Dominios que corresponden a las expresiones del algebra
relacional */

/* ------------------------------------------------------- */
/* Dominios correspondientes a los atributos de expresiones */

global domains
  listainteger     = integer*.
  listareal        = real*.
  listastring      = string*.
  listasymbol      = symbol*.
  listalistastring= listastring*.

global domains
  /* En seguida se define la informacion de un esquema relacional
*/
  campo        = c(string,integer,real).
  esquema      = campo*.
  listaesquema= esquema*.

  /* A continuacion se define un 'profile' mas completo */
  /* Nombre_rel,Esquema,Cualidad,Card,Size,Site,Sort,Indices */
  profile   = p(string,esquema,condiciones,
                real,integer,listainteger,string,listastring).
```

```
  /* Se definen ahora atributos de relaciones con niveles de
detalle */
  lugar = secuencia; disco.
          /* Esquema,Cualidad,Reduccion */
  atrib = a0(esquema,condiciones);
          /* Esquema,Cualidad.Reduccion,Card,Size */
          a1(esquema,condiciones,reduc,real,integer);
          /* E,C,R.N,S,Site,Sort,Indices,Lugar,Costo */
          a2(esquema,condiciones,reduc,
             real,integer,integer.string,listastring,lugar,real);
          /* Site,Sort,Indices,Lugar */
          a3(integer,string,listastring,lugar);
          /* Esquema,Card,Size */
          a4(esquema,real,integer);
          /* Esquema,Card */
          a5(esquema,real).
  atribs= atrib*.
  reduc = r(condiciones,atribs).

/*
-----------------------------------------------------------------
*/
/* Dominios correspondientes a expresiones y operadores
relacionales */

global domains
  /* Dominios correspondientes a operadores */
  nomoperador  = sl; pj; pd; in; un; sj;
                 scan(string); store;
                 index(string); sort(string); ship(integer).
  nomoperadores= nomoperador*.
  basemetodo   = simple; indice; sort; loop; merge.
  metoperador  = m(basemetodo,integer).
  elem         = v(string); c(string).
  comparador   = ma; ig; me.
  condicion    = cn(comparador,elem,elem).
  condiciones  = condicion*.
  remplazo     = r(string,string).
  remplazos    = remplazo*.
  atroperador  = nulo; s(condiciones); p(listastring);
                 u(remplazos,remplazos); j(condicion).
  operador     = o(nomoperador,metoperador,atroperador).

  listanomoperadores= nomoperadores*.

  /* Dominios correspondientes a expresiones */
  argumentos= expresion*.
  expresion = rel(profile); exp(operador,argumentos); vacia.

/* ------------------------------------------------------------ */
/* Dominios para el almacenamiento de subexpresiones            */
global domains
  subexpresion= rel(profile); exp(operador,listasymbol); vacia.
```

```
/* ---------------------------------------------------------- */
/* Dominios para los nodos de la busqueda                     */
global domains
  nodo = n(real,expresion).
  nodos= nodo*.

/* ---------------------------------------------------------- */
/* Archivo de salida de resultados                            */
global domains
  file = out1.



/*
============================================================================
*/
/* Area de datos
     */

database
  contador(symbol,integer).
  mesh(symbol,atrib,subexpresion).

  variable(char,expresion).
  cerrado(expresion).
  abierto(nodos).
  limite(real).
  mejor(nodo).

  numclases(integer).
  clases(integer).
  clase_var(integer,string).
  clase_cte(integer,string).
  mayor(integer,integer).

  selec(condicion).
  semijunta(condicion).
  junta(condicion).



/* ========================================================== */
/* Predicados globales                                        */

global predicates
  determ sitio_usuario(integer) - (o).
  determ hill_climbing(real) - (o).
  determ inversion_opt(real) - (o).
  determ tiempo_acceso(real) - (o).
  determ bytes_por_bloque(integer) - (o).
  determ factor_transmision(real) - (o).
  relacion(profile) - (o).
  dom(string,real) - (i,o).
  determ inicializa.
```

```
  determ abrearch(file,string) - (i,i).
  determ gensym(symbol,symbol) - (i,o).
  determ reset_gensym(symbol) - (i).
  generaint(integer) - (o).
  determ putlast(nomoperador,nomoperadores,nomoperadores) -
(i,i,o).
  determ append(esquema,esquema,esquema) - (i,i,o).
  determ append(condiciones,condiciones,condiciones) - (i,i,o).
  determ append(nomoperadores,nomoperadores,nomoperadores) -
(i,i,o).
  determ append(listastring,listastring,listastring) - (i,i,o).
  determ append(listaesquema,listaesquema,listaesquema) -
(i,i,o).
  determ append(atribs,atribs,atribs) - (i,i,o).
  determ normaliza_conjunto(listastring,listastring) - (i,o).
  determ normaliza_conjunto(condiciones,condiciones) - (i,o).
  determ conjuntos_iguales(listastring,listastring) - (i,i).
  determ conjuntos_iguales(condiciones,condiciones) - (i,i).
  determ subconjunto(listastring,listastring) - (i,i).
  determ subconjunto(condiciones,condiciones) - (i,i).
  determ incl(string,listastring,listastring) - (i,i,o).
  determ incl(integer,listainteger,listainteger) - (i,i,o).
  determ incl(esquema,listaesquema,listaesquema) - (i,i,o).
  determ incl(condicion,condiciones,condiciones) - (i,i,o).
  determ excl(string,listastring,listastring) - (i,i,o).
  determ excl(condicion,condiciones,condiciones) - (i,i,o).
  determ union(listastring,listastring,listastring) - (i,i,o).
  determ union(condiciones,condiciones,condiciones) - (i,i,o).
  determ union(listaesquema,listaesquema,listaesquema) - (i,i,o).
  determ inter(listastring,listastring,listastring) - (i,i,o)
(i,i,i).
  determ inter(condiciones,condiciones,condiciones) - (i,i,o)
(i,i,i).
  determ diferencia(listastring,listastring,listastring) -
(i,i,o).
  determ diferencia(condiciones,condiciones,condiciones) -
(i,i,o).
  determ length(listainteger,integer) - (i,o).
  determ length(esquema,integer) - (i,o).
  determ count(integer,listainteger,integer) - (i,i,o) (i,i,i).
  determ une_reduc(reduc,reduc,reduc) - (i,i,o).
  determ desigual(condicion,condicion) - (i,i).
  determ ceil(real,real) - (i,o).
  determ min(real,real,real) - (i,i,o).
  determ max(real,real,real) - (i,i,o).
  determ sum(listareal,real) - (i,o).
  determ sum(listainteger,integer) - (i,o).
  determ times(listareal,real) - (i,o).
  determ times(listainteger,integer) - (i,o).
  determ dom_campo(string,real) - (i,o).
  repeat.

  determ descompone(string,string,string) - (i,o,o) (i,o,i).
  determ compone(string,string,string) - (i,i,o).
  determ string_sust(remplazos,string,string) - (i,i,o).
```

```
  determ remplaza_campo(remplazos,string,string) - (i,i,o).
  determ remplaza_campos(remplazos,listastring,listastring) -
(i,i,o).
  determ remplaza_campoe(remplazos,campo,campo) - (i,i,c).
  determ remplaza_esquema(remplazos,esquema,esquema) - (i,i,o).
  determ remplaza_elemento(remplazos,elem,elem) - (i,i,o).
  determ remplaza_condicion(remplazos,condicion,condicion) -
(i,i,c).
  determ remplaza_conciciones(remplazos,condiciones,condiciones)
- (i,i,o).
  determ remplaza_profile(remplazos,profile,profile) - (i,i,o).
  determ remplaza_remplazos(remplazos,remplazos,remplazos) -
(i,i,o).
  determ remplaza_expresion(remplazos,expresion,expresion) -
(i,i,o).
  determ elemento_campos(elem,listastring) - (i,o).
  determ condicion_campos(condicion,listastring) - (i,o).
  determ condiciones_campos(condiciones,listastring) - (i,o).
  determ invierte_remplazos(remplazos,remplazos) - (i,o).
  determ esquema_campos(esquema,listastring) - (i,o).
  determ esquemas_campos(listaesquema,listalistastring) - (i,o).

  determ asigna(char,expresion) - (i,i).
  determ rel_profile(string,string,profile) - (i,o,o).
  determ parse(string,expresion) - (i,o).
  determ hoja(string,string,expresion) - (i,i,o).

  determ print(expresion) - (i).
  determ print3(condiciones) - (i).
  determ print3_1(condicion) - (i).
  determ print4(listastring) - (i).

  determ normaliza_simple(condiciones,listalistastring,
                          condiciones,condiciones,condiciones) -
(i,i,o,o,o).
  determ normaliza(listastring,condiciones,condiciones,
                   condiciones) - (i,i,i,o).
  determ es_aciclica(reduc) - (i).
  determ estima_selectividad(string,reduc,reduc,real) -
(i,i,i,o).
  determ aplica_sel(string,esquema,real,real,esquema,real) -
(i,i,i,i,o,o).

  metodo(nomoperador,metoperador) - (i,o).
  determ un_metodo(nomoperador,metoperador) - (i,o).
  determ programa_costea(nomoperadores,atrib,nomoperadores,real)
- (i,i,o,o).
  determ programa_ejecucion(symbol) - (i).
  condicion_subcon_campos(condiciones,listastring,condicion) -
(i,i,o).
  condicion_inters_campos(condiciones,listastring,condicion) -
(i,i,o).
  determ pgm(operador,atribs,
             atrib,listanomoperadores,condiciones) - (i,i,o,o,o).
  determ base0(expresion,atrib) - (i,o).
```

```
  determ ev0(operador,atribs,atrib) - (i,i,o).
  determ base1(expresion,atrib) - (i,o).
  determ ev1(operador,atribs,atrib) - (i,i,o).
  determ base2(expresion,atrib) - (i,o).
  determ ev2(operador,atribs,atrib) - (i,i,o).
  determ eval(string,string,expresion,atrib) - (i,i,i,o).
  determ call0(string,expresion,atrib) - (i,i,o).
  determ call1(string,operador,atribs,atrib) - (i,i,i,o).

  determ empaca(expresion,symbol) - (i,o).
  determ desempaca(symbol,expresion) - (i,o).
  determ destruye_mesh.
  determ exp_costo(expresion,real) - (i,o).
  determ id_costo(symbol,real) - (i,o).

  determ divseleccion(condiciones,listastring,condiciones,
                      condiciones) - (i,i,o,o).
  genera(integer,expresion,expresion) - (i,i,o).
  determ proc_inicial(integer,expresion,expresion) - (i,i,o).

  determ search(listainteger,expresion,expresion) - (i,i,o).


/* ========================================================== */
/* Rutinas de uso comun con patron de entrada compuesto */

predicates
  select(atrib,atribs).
  select(expresion,argumentos).
  select(campo,esquema).
  select(integer,listainteger).
  select(condicion,condiciones).
  select(remplazo,remplazos).
  select(listastring,listalistastring).
  select(esquema,listaesquema).
  select(string,listastring).
  select(symbol,listasymbol).
  select(nomoperador,nomoperadores).
  select2(esquema,campo,esquema).
  select2(atribs,atrib,atribs).
  select2(listalistastring,listastring,listalistastring).
  select2(listainteger,integer,listainteger).
  select2(condiciones,condicion,condiciones).
  select2(listastring,string,listastring).

clauses
  select(A,[A|_]).
  select(A,[_|X]):- select(A,X).

  select2([A|X],A,X).
  select2([A|X],B,[A|Y]):- select2(X,B,Y).
```

```
/* parametros.pro */

project "tesis2"
include "dominios.pro"


/* ========================================================== */
/* parametros de operacion del sistema                        */

clauses
  sitio_usuario(2).
  hill_climbing(1.01).
  inversion_opt(0.1).
  tiempo_acceso(3E-3).
  bytes_por_bloque(512).
  factor_transmision(1000).

clauses
  /* esquema relacional 1 */
  relacion(p("t.books",[c("t.title",15,1000000),
                        c("t.author",15,800000),
                        c("t.pname",15,500),
                        c("t.lc_no",6,1000000)],
           [],
           1000000,51,[1],"?",["t.lc_no","t.author"])).
  relacion(p("t.publishers",[c("t.pname",15,500),
                             c("t.paddr",15,500),
                             c("t.pcity",10,50)],
           [],
           500,40,[1],"?",["t.pname","t.pcity"])).
  relacion(p("t.borrowers",[c("t.name",15,40000),
                            c("t.addr",15,30000),
                            c("t.city",10,50),
                            c("t.card_no",6,40000)],
           [],
           40000,46,[1],"?",["t.card_no","t.city"])).
  relacion(p("t.loans",[c("t.card_no",6,20000),
                        c("t.lc_no",6,40000),
                        c("t.date",6,300)],
           [],
           40000,18,[2],date,["t.card_no","t.lc_no"])).

  /* esquema relacional 2 */
  relacion(p("t.r1",[c("t.c1",10,2000),
                     c("t.c2",20,1000),
                     c("t.c3",20,150)],
           [],
           2000,50,[1],"?",[])).
  relacion(p("t.r2",[c("t.c1",10,2000),
                     c("t.c2",20,1200),
                     c("t.c3",20,450)],
           [],
           5000,50,[2],"?",[])).
  relacion(p("t.r3",[c("t.c1",10,1500),
                     c("t.c2",20,900),
```

```
                    c("t.c3",20,150)],
            [],
            2500,50,[3],"?",[])).

/* esquema relacional 3 */
relacion(p("t.s",[c("t.s",10,10000),
                  c("t.name",10,10000),
                  c("t.location",10,50)],
            [],
            10000,30,[1],"?",[])).
relacion(p("t.y",[c("t.s",10,1000),
                  c("t.p",10,1000)],
            [],
            100000,20,[2],"?",[])).
relacion(p("t.p",[c("t.p",10,10000),
                  c("t.name",10,10000),
                  c("t.type",10,5)],
            [],
            10000,30,[3],"?",[])).

/* esquema relacional 3 */
relacion(p("t.a1",[c("t.a",1,900),
                   c("t.b",1,900),
                   c("t.c",1,900)],
            [],
            1000000,3,[1],"?",[])).
relacion(p("t.a2",[c("t.a",1,800),
                   c("t.b',1,800),
                   c("t.c",1,800)],
            [],
            900000,3,[2],"?",[])).
relacion(p("t.a3",[c("t.a",1,700),
                   c("t.b",1,700),
                   c("t.c",1,700)],
            [],
            900000,3,[3],"?",[])).
relacion(p("t.a4",[c("t.a",1,500),
                   c("t.b",1,500),
                   c("t.c",1,500)],
            [],
            800000,3,[4],"?",[])).
relacion(p("t.a5",[c("t.a",1,600),
                   c("t.b",1,600),
                   c("t.c',1,600)],
            [],
            800000,3,[5],"?",[])).
relacion(p("t.a6",[c("t.a",1,500),
                   c("t.b",1,500),
                   c("t.c",1,500)],
            [],
            800000,3,[6],"?",[])).
relacion(p("t.a7",[c("t.a",1,300),
                   c("t.b",1,300),
                   c("t.c",1,300)],
            [],
```

```
                600000,3,[7],"?",[])).

    /* esquema relacional 4 */
    relacion(p("t.books1",[c("t.title",15,500000),
                           c("t.author",15,400000),
                           c("t.pname",15,500),
                           c("t.lc_no",6,500000),
                           c("t.library",1,3)],
                [cn(me,v("t.library"),c("'4'"))],
                500000,52,[1],"?",["t.lc_no","t.author"])).
    relacion(p("t.books2",[c("t.title",15,300000),
                           c("t.author",15,200000),
                           c("t.pname",15,500),
                           c("t.lc_no",6,300000),
                           c("t.library",1,1)],
                [cn(ig,v("t.library"),c("'4'"))],
                300000,52,[2],"?",["t.lc_no","t.author"])).
    relacion(p("t.books3",[c("t.title",15,200000),
                           c("t.author",15,150000),
                           c("t.pname",15,500),
                           c("t.lc_no",6,200000),
                           c("t.library",1,1)],
                [cn(ig,v("t.library"),c("'5'"))],
                200000,52,[3],"?",["t.lc_no","t.author"])).

clauses
    dom(lc_no,1000000).
    dom(pname,500).
    dom(card_no,40000).
    dom(s,10000).
    dom(p,10000).
    dom(location,50).
    dom(type,5).
    dom(c1,5000).
    dom(c2,1500).
    dom(c3,1500).
    dom(a,1000).
    dom(b,1000).
    dom(c,1000).

predicates
    e0(expresion).
    e1(expresion).
    e2(expresion).
    e3(expresion).
    e4(expresion).
    e5(expresion).

clauses
    e0(exp(o(pd,Met,nulo),[E1,E2])):-
        un_metodo(pd,Met),
        hoja(loans,l,E1), hoja(books,b,E2).
    e1(exp(o(pd,Met,nulo),[E1,E2])):-
        un_metodo(pd,Met),
        e0(E1), hoja(borrowers,w,E2).
```

```
  e2(exp(o(s1,Met,s([cn(ig,v("b.lc_no"),v("1.lc_no")),
                     cn(ig,v("w.card_no"),v("1.card_no"))])),
         [E1])):-
    un_metodo(s1,Met),
    e1(E1).
  e3(exp(o(pj,Met,p(["b.title","b.author","b.lc_no",
                     "w.name","w.addr",
                     "w.city","w.card_no",
                     "1.date"])),
         [E1])):-
    un_metodo(pj,Met),
    e2(E1).
  e4(exp(o(s1,Met,s([cn(me,v("1.date"),c("'010182'"))])),[E1])):-
    un_metodo(s1,Met),
    e3(E1).
  e5(exp(o(pj,Met,p(["b.title"])),[E1])):-
    un_metodo(pj,Met),
    e4(E1).

inicializa:- e5(X1), asigna('w',X1), fail.
inicializa.
```

```
/* utilerias.pro */

project "tesis2"
include "dominios.pro"


/* ========================================== */
/* Implementaciones de operaciones relacionales */


/* ========================================================
*/
/* Predicados auxiliares
*/

predicates
  normal_es(atribs,atribs).

clauses
  abrearch(F,S):- openappend(F,S), !, writedevice(F).
  abrearch(F,S):- openwrite(F,S), writedevice(F).

  gensym(X,S):- retract(contador(X,M)), !, N= M+1,
                asserta(contador(X,N)), str_int(S1,N),
                concat(X,S1,S).
  gensym(X,S):- asserta(contador(X,0)), concat(X,"0",S).

  reset_gensym(X):- retract(contador(X,_)), fail.
  reset_gensym(_).

  generaint(1).
  generaint(N):- generaint(N1), N= N1+1.

  putlast(A,[],[A]).
  putlast(A,[B|X],[B|Y]):- putlast(A,X,Y).

  append([],X,X).
  append([A|X],Y,[A|Z]):- append(X,Y,Z).

  normaliza_conjunto([],[]).
  normaliza_conjunto([A|X],Y):- select(A,X), !,
normaliza_conjunto(X,Y).
  normaliza_conjunto([A|X],[A|Y]):- normaliza_conjunto(X,Y).

  conjuntos_iguales([],[]).
  conjuntos_iguales([A|X],Y):- select2(Y,A,Z), !,
conjuntos_iguales(X,Z).

  subconjunto([],_).
  subconjunto([A|X],Y):- select(A,Y), !, subconjunto(X,Y).

  incl(A,X,X):- select(A,X), !.
  incl(A,X,[A|X]).

  excl(A,X,Y):- select2(X,A,Y), !.
```

```
excl(_,X,X).

union([],Y,Y).
union([A|X],Y,Z):- incl(A,Y,Y1), union(X,Y1,Z).

inter([],_,[]).
inter([A|X],Y,[A|Z]):- select(A,Y), !, inter(X,Y,Z).
inter([_|X],Y,Z):- inter(X,Y,Z).

diferencia(X,[],X).
diferencia(X,[A|Y],Z):- excl(A,X,X1), diferencia(X1,Y,Z).

une_reduc(r(Cn1,Es1),r(Cn2,Es2),r(Cn3,Es3)):-
        union(Cn1,Cn2,Cn3),
        append(Es1,Es2,Esk),
        normal_es(Esk,Es3).

normal_es([],[]).
normal_es([a5(E1,N1)|X],Y):-
        select(c(C,_,_),E1),
        select2(X,a5(E2,_),X2),
        select(c(C,_,_),E2), !,
        normal_es([a5(E1,N1)|X2],Y).
normal_es([A|X],[A|Y]):-
        normal_es(X,Y).

length([],0).
length([_|X],N):- length(X,N1), N= N1+1.

count(_,[],0).
count(A,[A|X],N):- !, count(A,X,N1), N= N1+1.
count(A,[_|X],N):- count(A,X,N).

desigual(X,X):- !, fail.
desigual(_,_).

ceil(N,N):- 32000<N, !.
ceil(N,N):- N=round(N), !.
ceil(N,M):- M= round(N-0.5)+1.

min(A,B,A):- A<B, !.
min(_,B,B).

max(A,B,A):- B<A, !.
max(_,B,B).

sum([],0).
sum([A|X],N):- sum(X,N1), N= N1+A.

times([],1).
times([A|X],N):- times(X,N1), N= N1*A.

repeat.
repeat:- repeat.
```

```
/* ============================================================
*/
/* Predicados auxiliares para manejo de campos
*/

clauses
  descompone(S,S1,S2):- fronttoken(S,S1,R1),
frontchar(R1,'.',S2).
  compone(S1,S2,S):- frontchar(R1,'.',S2), fronttoken(S,S1,R1).

  string_sust(Rs,S1,S2):- select(r(S1,S2),Rs), !.
  string_sust(_,S,S).

  remplaza_campo(Rs,C1,C2):- descompone(C1,S1,S2),
                             string_sust(Rs,S1,S3),
                             compone(S3,S2,C2), !.
  remplaza_campo(_,C,C).

  remplaza_campos(_,[],[]).
  remplaza_campos(Rs,[C1|C1s],[C2|C2s]):-
                remplaza_campo(Rs,C1,C2),
                remplaza_campos(Rs,C1s,C2s).

  remplaza_campoe(Rs,c(C1,I,R),c(C2,I,R)):-
                remplaza_campo(Rs,C1,C2).

  remplaza_esquema(_,[],[]).
  remplaza_esquema(Rs,[C1|C1s],[C2|C2s]):-
                 remplaza_campoe(Rs,C1,C2),
                 remplaza_esquema(Rs,C1s,C2s).

  remplaza_elemento(_,c(X),c(X)).
  remplaza_elemento(Rs,v(X),v(Y)):- remplaza_campo(Rs,X,Y).

  remplaza_condicion(Rs,cn(Op,E1,E2),cn(Op,E3,E4)):-
                   remplaza_elemento(Rs,E1,E3),
                   remplaza_elemento(Rs,E2,E4).

  remplaza_condiciones(_,[],[]).
  remplaza_condiciones(Rs,[C1|C1s],[C2|C2s]):-
                     remplaza_condicion(Rs,C1,C2),
                     remplaza_condiciones(Rs,C1s,C2s).

  remplaza_profile(Rs,p(Nombre1,Esquema1,Cualidad1,
                        Card,Size,Site,Sort1,Indices1),
                      p(Nombre2,Esquema2,Cualidad2,
                        Card,Size,Site,Sort2,Indices2)):-
                 remplaza_campo(Rs,Nombre1,Nombre2),
                 remplaza_esquema(Rs,Esquema1,Esquema2),
                 remplaza_condiciones(Rs,Cualidad1,Cualidad2),
                 remplaza_campo(Rs,Sort1,Sort2),
                 remplaza_campos(Rs,Indices1,Indices2).
```

```
remplaza_remplazos(_,[],[]).
remplaza_remplazos(Rs,[r(S1,S2)|R1],[r(S1,S3)|R2]):-
                    string_sust(Rs,S2,S3),
                    remplaza_remplazos(Rs,R1,R2).

remplaza_expresion(_,vacia,vacia).
remplaza_expresion(Rs,rel(Profile1),rel(Profile2)):-
                    remplaza_profile(Rs,Profile1,Profile2).
remplaza_expresion(Rs,exp(o(s1,Met1,s(Arg1)),[E1]),
                        exp(o(sl,Met1,s(Arg2)),[E2])):-
                    remplaza_condiciones(Rs,Arg1,Arg2),
                    remplaza_expresion(Rs,E1,E2).
remplaza_expresion(Rs,exp(o(pj,Met1,p(Arg1)),[E1]),
                        exp(o(pj,Met1,p(Arg2)),[E2])):-
                    remplaza_campos(Rs,Arg1,Arg2),
                    remplaza_expresion(Rs,E1,E2).
remplaza_expresion(Rs,exp(o(pd,Met1,nulo),[E1,E2]),
                        exp(o(pd,Met1,nulo),[E3,E4])):-
                    remplaza_expresion(Rs,E1,E3),
                    remplaza_expresion(Rs,E2,E4).
remplaza_expresion(Rs,exp(o(jn,Met1,s(Arg1)),[E1,E2]),
                        exp(o(jn,Met1,s(Arg2)),[E3,E4])):-
                    remplaza_condiciones(Rs,Arg1,Arg2),
                    remplaza_expresion(Rs,E1,E3),
                    remplaza_expresion(Rs,E2,E4).
remplaza_expresion(Rs,exp(o(sj,Met1,s(Arg1)),[E1,E2]),
                        exp(o(sj,Met1,s(Arg2)),[E3,E4])):-
                    remplaza_condiciones(Rs,Arg1,Arg2),
                    remplaza_expresion(Rs,E1,E3),
                    remplaza_expresion(Rs,E2,E4).
remplaza_expresion(Rs,exp(o(un,Met1,u(Arg1,Arg2)),[E1,E2]),
                        exp(o(un,Met1,u(Arg3,Arg4)),[E3,E4])):-
                    remplaza_remplazos(Rs,Arg1,Arg3),
                    remplaza_remplazos(Rs,Arg2,Arg4),
                    remplaza_expresion(Rs,E1,E3),
                    remplaza_expresion(Rs,E2,E4).

elemento_campos(c(_),[]).
elemento_campos(v(C),[C]).
condicion_campos(cn(_,E1,E2),Cs):- elemento_campos(E1,C1),
                                   elemento_campos(E2,C2),
                                   union(C1,C2,Cs).
condiciones_campos([],[]).
condiciones_campos([Cn|Cns],Cs):- condicion_campos(Cn,C1),
                                  condiciones_campos(Cns,C2),
                                  union(C1,C2,Cs).

invierte_remplazos([],[]).
invierte_remplazos([r(S1,S2)|R1s],[r(S2,S1)|R2s]):-
                    invierte_remplazos(R1s,R2s).

esquema_campos(Es,Cp):- findall(X,select(c(X,_,_),Es),Cp).
esquemas_campos([],[]).
esquemas_campos([Es|Ess],[Cp|Cps]):-
                esquema_campos(Es,CP),
```

```
esquemas_campos(Ess,Cps).

dom_campo(C,D):- descompone(C,_,N), dom(N,D).
```

```
/* parser.pro */

project "tesis2"
include "dominios.pro"


/* ============================================================ */
/* Esquemas relacionales ejemplo y construccion de expresiones */

clauses
  rel_profile(Nombre,Var,p(Nom,Esq,Cual,Card,Size,Site,Sort,Indices)):
              relacion(X),
X=p(Nom,Esq,Cual,Card,Size,Site,Sort,Indices),
              descompone(Nom,Var,Nombre), !.


/* ============================================================ */
/* Manejo de variables                                          */

clauses
  asigna(X,_):- retract(variable(X,_)), fail.
  asigna(X,E):- asserta(variable(X,E)).



/* ============================================================ */
/* Construccion de estructuras y parse de strings               */

clauses
  hoja(Nombre_relacion,Nombre_variable,rel(Profile2)):-
      rel_profile(Nombre_relacion,Nom,Profile),
      remplaza_profile([r(Nom,Nombre_Variable)],Profile,Profile2).

predicates
  parse_expresion(string,expresion,string).
  parse_condiciones(string,condiciones,string).
  parse_campos(string,listastring,string).
  parse_comparador(string,comparador,string).
  parse_elem(string,elem,string).
  parse_remplazos(string,remplazos,string).
  token_inicial(string,string,string).
  token_inicial1(string,string,string,string).
  token_inicial2(string,string,string).
  empty_string(string).

clauses
  parse(S1,E1):- parse_expresion(S1,E1,S2), empty_string(S2).

  parse_expresion(S1,exp(o(sl,Met,s(Cond)),[E1]),S2):-
                  token_inicial(S1,sl,S3), !,
                  un_metodo(sl,Met),
                  token_inicial(S3,":",S4),
                  parse_condiciones(S4,Cond,S5),
                  token_inicial(S5,":",S6),
                  parse_expresion(S6,E1,S2).
```

```
parse_expresion(S1,exp(o(pj,Met,p(Camp)),[E1]),S2):-
                token_inicial(S1,pj,S3), !,
                un_metodo(pj,Met),
                token_inicial(S3,":",S4),
                parse_campos(S4,Camp,S5),
                token_inicial(S5,":",S6),
                parse_expresion(S6,E1,S2).
parse_expresion(S1,exp(o(pd,Met,nulo),[E1,E2]),S2):-
                token_inicial(S1,pd,S3), !,
                un_metodo(pd,Met),
                token_inicial(S3,":",S4),
                parse_expresion(S4,E1,S5),
                token_inicial(S5,",",S6),
                parse_expresion(S6,E2,S2).
parse_expresion(S1,exp(o(jn,Met,s(Cond)),[E1,E2]),S2):-
                token_inicial(S1,jn,S3), !,
                un_metodo(jn,Met),
                token_inicial(S3,":",S4),
                parse_condiciones(S4,Cond,S5),
                token_inicial(S5,":",S6),
                parse_expresion(S6,E1,S7),
                token_inicial(S7,",",S8),
                parse_expresion(S8,E2,S2).
parse_expresion(S1,exp(o(un,Met,u(Remp1,Remp2)),[E1,E2]),S2):
                token_inicial(S1,un,S3), !,
                un_metodo(un,Met),
                token_inicial(S3,":",S4),
                token_inicial(S4,"(",T1),
                parse_remplazos(T1,Remp1,S5),
                token_inicial(S5,")",T2),
                token_inicial(T2,"(",T3),
                parse_remplazos(T3,Remp2,S6),
                token_inicial(S6,")",T4),
                token_inicial(T4,":",S7),
                parse_expresion(S7,E1,S8),
                token_inicial(S8,",",S9),
                parse_expresion(S9,E2,S2).
parse_expresion(S1,E1,S2):-
                token_inicial(S1,"(",S3), !,
                parse_expresion(S3,E1,S4),
                token_inicial(S4,")",S2).
parse_expresion(S1,E1,S2):-
                token_inicial(S1,"%",S3), !,
                frontchar(S3,Var,S2),
                variable(Var,E1), !.
parse_expresion(S1,E1,S2):-
                token_inicial(S1,Var,S3),
                isname(Var),
                token_inicial(S3,"-",S4),
                token_inicial(S4,Rel,S2),
                hoja(Rel,Var,E1).

parse_condiciones(S1,[cn(Cmp,C1,C2)|Cond],S2):-
                  token_inicial(S1,"(",S3),
                  parse_elem(S3,C1,S4),
```

```
                    parse_comparador(S4,Cmp,S5),
                    parse_elem(S5,C2,S6),
                    token_inicial(S6,")",S7), !,
                    parse_condiciones(S7,Cond,S2).
parse_condiciones(S1,[],S1).

parse_campos(S1,[C1|Camp],S2):-
            parse_elem(S1,v(C1),S3), !,
            parse_campos(S3,Camp,S2).
parse_campos(S1,[],S1).

parse_elem(S1,v(C1),S2):- token_inicial(S1,C1,S2),
                          fronttoken(C1,Cc,_),
                          isname(Cc), !.
parse_elem(S1,c(C1),S2):- token_inicial(S1,C1,S2),
                          frontchar(C1,'''',_).

parse_comparador(S1,ig,S2):- token_inicial(S1,"=",S2), !.
parse_comparador(S1,ma,S2):- token_inicial(S1,">",S2), !.
parse_comparador(S1,me,S2):- token_inicial(S1,"<",S2).

parse_remplazos(S1,[],S1):-
              token_inicial(S1,")",_), !.
parse_remplazos(S1,[r(Sr1,Sr2)],S2):-
              token_inicial(S1,Sr1,S3),
              token_inicial(S3,"-",S4),
              token_inicial(S4,Sr2,S2),
              token_inicial(S2,")",_), !.
parse_remplazos(S1,[r(Sr1,Sr2)|Rs],S2):-
              token_inicial(S1,Sr1,S3),
              token_inicial(S3,"-",S4),
              token_inicial(S4,Sr2,S5),
              token_inicial(S5,",",S6),
              parse_remplazos(S6,Rs,S2).

token_inicial(S1,Tk,S2):- fronttoken(S1,T0,S0),
                          token_inicial1(T0,S0,Tk,S2).
token_inicial1("'",S0,Tk,S2):- token_inicial2(S0,T0,S2), !,
                               frontchar(Tk,'''',T0).
token_inicial1(T0,S0,Tk,S2):- fronttoken(S0,".",S1), !,
concat(T0,".",T1),
                              token_inicial(S1,Tk1,S2),
concat(T1,Tk1,Tk).
token_inicial1(T0,S0,Tk,S2):- fronttoken(S0,"_",S1), !,
concat(T0,"_",T1),
                              token_inicial(S1,Tk1,S2),
concat(T1,Tk1,Tk).
token_inicial1(Tk,S2,Tk,S2).
token_inicial2(S1,"'",S2):- frontchar(S1,'''',S2), !.
token_inicial2(S1,Tk,S2):- frontchar(S1,Ch,S0),
token_inicial2(S0,T0,S2),
                           frontchar(Tk,Ch,T0).

empty_string(S):- not(fronttoken(S,_,_)).
```

```
/* print.pro */

project "tesis2"
include "dominios.pro"


/* ============================================================
*/
/* Predicados de impresion
*/

predicates
  print2(expresion,integer).
  print3_2(elem).
  print5(remplazos).
  spc(integer).

clauses
  print(Expresion):- print2(Expresion,0).

  print2(vacia,Nivel):-
        spc(Nivel), write(vacia), nl.
  print2(rel(p(Nombre,_,Cual,_,_,_,_,_)),Nivel):-
        descompone(Nombre,Var,Rel),
        spc(Nivel), write(Var," - ",Rel), write(" : "),
print3(Cual), nl.
  print2(exp(o(sl,m(Met,_),s(Condiciones)),[E1]),Nivel):-
        spc(Nivel), write("sl ",Met," : "), print3(Condiciones),
nl,
        Nivel1= Nivel+1, print2(E1,Nivel1).
  print2(exp(o(pj,m(Met,_),p(Nomcampos)),[E1]),Nivel):-
        spc(Nivel), write("pj ",Met," : "), print4(Nomcampos),
nl,
        Nivel1= Nivel+1, print2(E1,Nivel1).
  print2(exp(o(pd,m(Met,N),nulo),[E1,E2]),Nivel):-
        spc(Nivel), write("pd ",Met," ",N), nl,
        Nivel1= Nivel+1, print2(E1,Nivel1), print2(E2,Nivel1).
  print2(exp(o(jn,m(Met,N),s(Condiciones)),[E1,E2]),Nivel):-
        spc(Nivel), write("jn ",Met," ",N," : "),
print3(Condiciones), nl,
        Nivel1= Nivel+1, print2(E1,Nivel1), print2(E2,Nivel1).
  print2(exp(o(sj,m(Met,_),j(Condicion)),[E1,E2]),Nivel):-
        spc(Nivel), write("sj ",Met," : "), print3_1(Condicion),
nl,
        Nivel1= Nivel+1, print2(E1,Nivel1), print2(E2,Nivel1).
  print2(exp(o(un,m(Met,N),u(Remp1,Remp2)),[E1,E2]),Nivel):-
        spc(Nivel), write("un ",Met," ",N," : "),
        write('('), print5(Remp1), write(") ("),
        print5(Remp2), write(')'), nl,
        Nivel1= Nivel+1, print2(E1,Nivel1), print2(E2,Nivel1).

  print3([]).
  print3([Condicion]):- !, print3_1(Condicion).
  print3([Condicion|Condiciones]):-
        print3_1(Condicion), write(" AND "), print3(Condiciones).
```

```
print3_1(cn(ma,A,B)):-
        print3_2(A), write(">"), print3_2(B).
print3_1(cn(ig,A,B)):-
        print3_2(A), write("="), print3_2(B).
print3_1(cn(me,A,B)):-
        print3_2(A), write("<"), print3_2(B).

print3_2(v(Campo)):- write(Campo).
print3_2(c(Constante)):- write(Constante).

print4([]).
print4([Campo]):- !, write(Campo).
print4([Campo|Campos]):-
      write(Campo), write(","), print4(Campos).

print5([]).
print5([r(S1,S2)]):- !, write(S1,"->",S2).
print5([r(S1,S2)|Rs]):-
      write(S1,"->",S2,","), print5(Rs).

spc(N):- N<=0, !.
spc(N):- generaint(N1), write("   "), N1=N, !.
```

```
/* analisis.pro */

code= 2000

project "tesis2"
include "dominios.pro"


/* ========================================== */
/* Predicados para el analisis de condiciones */

predicates
  construye_modelo(condiciones).
  destruye_modelo.
  agrega_condiciones(condiciones).
  agrega_igual(elem,elem).
  agrega_mayor(elem,elem).
  en_clase(elem,integer).
  pon_clase(elem,integer).
  clasifica(elem,integer).
  merge_clases(integer,integer).
  asigna_mayor(integer,integer).
  derivable(condicion).
  es_igual(elem,elem).
  es_mayor(elem,elem).
  clase_mayor(integer,integer).
  grafica_aciclica(listainteger).
  hay_nodo_mayor(integer,listainteger).
  constantes_inconsistentes.
  agrega_arcos.
  elimina_arcos1.
  elimina_arcos2.
  num_caminos(integer,integer,integer).

clauses
  construye_modelo(Condiciones):-
                    asserta(numclases(0)),
                    agrega_condiciones(Condiciones),
                    findall(X,clases(X),L),
                    grafica_aciclica(L),
                    not(constantes_inconsistentes),
                    agrega_arcos,
                    elimina_arcos1,
                    elimina_arcos2, !.
  construye_modelo(_):- destruye_modelo, fail.

  destruye_modelo:- retract(numclases(_)), fail.
  destruye_modelo:- retract(clases(_)), fail.
  destruye_modelo:- retract(mayor(_,_)), fail.
  destruye_modelo:- retract(clase_cte(_,_)), fail.
  destruye_modelo:- retract(clase_var(_,_)), fail.
  destruye_modelo.

  agrega_condiciones([]).
  agrega_condiciones([cn(me,Elem1,Elem2)|Conds]):-
```

```
                    agrega_mayor(Elem2,Elem1),
                    agrega_condiciones(Conds).
  agrega_condiciones([cn(ig,Elem1,Elem2)|Conds]):-
                    agrega_igual(Elem2,Elem1),
                    agrega_condiciones(Conds).
  agrega_condiciones([cn(ma,Elem1,Elem2)|Conds]):-
                    agrega_mayor(Elem1,Elem2),
                    agrega_condiciones(Conds).

  agrega_igual(Elem1,Elem2):-
               en_clase(Elem1,N), en_clase(Elem2,M), !,
merge_clases(N,M);
               en_clase(Elem1,N), !, pon_clase(Elem2,N);
               en_clase(Elem2,N), !, pon_clase(Elem1,N);
               clasifica(Elem1,N), pon_clase(Elem2,N).
  agrega_mayor(Elem1,Elem2):-
               clasifica(Elem1,N1), clasifica(Elem2,N2),
               asigna_mayor(N1,N2).

  en_clase(v(Var),N):- clase_var(N,Var).
  en_clase(c(Cte),N):- clase_cte(N,Cte).

  pon_clase(v(Var),N):- asserta(clase_var(N,Var)).
  pon_clase(c(Cte),N):- not(clase_cte(N,_)),
asserta(clase_cte(N,Cte)).

  clasifica(v(Var),N):- clase_var(N,Var), !;
                        retract(numclases(M)), N= M+1, !,
                          asserta(clases(N)),
                          asserta(numclases(N)),
asserta(clase_var(N,Var)).
  clasifica(c(Cte),N):- clase_cte(N,Cte), !;
                        retract(numclases(M)), N= M+1, !,
                          asserta(clases(N)),
                          asserta(numclases(N)),
asserta(clase_cte(N,Cte)).

  merge_clases(N,N):- !.
  merge_clases(N,M):- retract(clases(M)),
                      retract(clase_var(M,Var)),
                      asserta(clase_var(N,Var)), fail.
  merge_clases(N,M):- retract(clase_cte(M,Cte)),
                      asserta(clase_cte(N,Cte)), fail.
  merge_clases(N,M):- retract(mayor(M,X)),
                      asigna_mayor(N,X), fail.
  merge_clases(N,M):- retract(mayor(X,M)),
                      asigna_mayor(X,N), fail.
  merge_clases(_,_).

  asigna_mayor(C1,C2):- mayor(C1,C2), !.
  asigna_mayor(C1,C2):- asserta(mayor(C1,C2)).

  derivable(cn(me,Elem1,Elem2)):- es_mayor(Elem2,Elem1).
  derivable(cn(ig,Elem1,Elem2)):- es_igual(Elem1,Elem2).
  derivable(cn(ma,Elem1,Elem2)):- es_mayor(Elem1,Elem2).
```

```
  es_igual(Elem1,Elem2):- en_clase(Elem1,N), en_clase(Elem2,N).
  es_mayor(Elem1,Elem2):- en_clase(Elem1,N1), en_clase(Elem2,N2),
                          clase_mayor(N1,N2), !.

  clase_mayor(N,M):- mayor(N,M).
  clase_mayor(N,M):- mayor(X,M), clase_mayor(N,X).

  grafica_aciclica([]).
  grafica_aciclica(Nodos):- select2(Nodos,Nodo,Resto),
                            not(hay_nodo_mayor(Nodo,Nodos)), !,
                            grafica_aciclica(Resto).

  hay_nodo_mayor(Nodo,Nodos):- mayor(X,Nodo), select(X,Nodos).

  constantes_inconsistentes:- clase_cte(N,CteN),
                              clase_cte(M,CteM),
                              not(CteN>CteM),
                              clase_mayor(N,M).

  agrega_arcos:- clase_cte(N,CteN),
                 clase_cte(M,CteM),
                 CteN>CteM,
                 asigna_mayor(N,M), fail.
  agrega_arcos.

  elimina_arcos1:- mayor(N,M),
                   not(num_caminos(N,M,1)),
                   retract(mayor(N,M)), fail.
  elimina_arcos1.
  elimina_arcos2:- mayor(N,M), clase_cte(N,_), clase_cte(M,_),
                   retract(mayor(N,M)), fail.
  elimina_arcos2.
  num_caminos(N,M,K):- findall(X,clase_mayor(X,M),L),
count(N,L,K).

predicates
  construye_predicado(listalistastring,condiciones,condiciones,
condiciones).
  destruye_predicados_auxiliares.
  predicado_clases(listalistastring).
  predicado_clase(integer,listalistastring).
  predicado_clase2(listastring,listalistastring).
  predicado_mayor(listalistastring).
  predicado_mayor2(integer,integer,listalistastring).

clauses
  construye_predicado(Esquema,CondJunta,CondSemijunta,
                      CondSeleccion):-
                     predicado_clases(Esquema),
                     predicado_mayor(Esquema),
                     findall(X,junta(X),CondJunta),
                     findall(X,semijunta(X),CondSemijunta),
                     findall(X,selec(X),CondSeleccion),
                     destruye_predicados_auxiliares.
```

```
  destruye_predicados_auxiliares:- retract(junta(_)), fail.
  destruye_predicados_auxiliares:- retract(semijunta(_)). fail.
  destruye_predicados_auxiliares:- retract(selec(_)), fail.
  destruye_predicados_auxiliares.

  predicado_clases(Esquemas):- clases(C1),
predicado_clase(C1,Esquemas).
  predicado_clases(_).

  predicado_clase(C1,_):- clase_cte(C1,Cte), !,
                          clase_var(C1,Var),
                          asserta(selec(cn(ig,v(Var),c(Cte)))),
                          fail.
  predicado_clase(C1,Esquemas):- findall(X,clase_var(C1,X),Vars),
                                 predicado_clase2(Vars,Esquemas).

  predicado_clase2([Var1|Vars],Esquemas):-
                  select(Esquema,Esquemas), select(Var1,Esquema),
                  select(Var2,Vars), select(Var2,Esquema), !,
                  asserta(selec(cn(ig,v(Var1),v(Var2)))),
                  predicado_clase2(Vars,Esquemas).
  predicado_clase2([Var1,Var2|Vars],Esquemas):-
                  asserta(junta(cn(ig,v(Var1),v(Var2)))),
                  predicado_clase2([Var2|Vars],Esquemas).

  predicado_mayor(Esquemas):- mayor(N,M),
                              predicado_mayor2(M,N,Esquemas),
fail.
  predicado_mayor(_).

  predicado_mayor2(C1,C2,_):-
                  clase_cte(C2,Cte), clase_var(C1,Var),
                  asserta(selec(cn(me,v(Var),c(Cte)))), !.
  predicado_mayor2(C1,C2,_):-
                  clase_cte(C1,Cte), clase_var(C2,Var),
                  asserta(selec(cn(ma,v(Var),c(Cte)))), !.
  predicado_mayor2(C1,C2,Esquemas):-
                  clase_var(C1,Var1),
                  select(Esquema,Esquemas),
                  select(Var1,Esquema),
                  clase_var(C2,Var2),
                  select(Var2,Esquema),
                  asserta(selec(cn(ma,v(Var2),v(Var1)))), !.
  predicado_mayor2(C1,C2,_):-
                  clase_var(C1,Var1), clase_var(C2,Var2),
                  asserta(junta(cn(ma,v(Var2),v(Var1)))), !.

predicates
  calcula_diferencia(condiciones,condiciones).

clauses
  normaliza_simple(Cs1,Lc,C1,C2,C3):- construye_modelo(Cs1),
                                      construye_predicado(Lc,C1,C2,C3),
!,
                                                destruye_modelo.
```

```
  normaliza(Esquema,Cualidad,Seleccion,SeleccionNormal):-
            append(Cualidad,Seleccion,C2),
            normaliza_simple(C2,[Esquema],_,_,Cualidad2),
            construye_modelo(Cualidad),
            calcula_diferencia(Cualidad2,SeleccionNormal),
            destruye_modelo.

  calcula_diferencia([],[]).
  calcula_diferencia([Cond|Conds],Normal):-
                     derivable(Cond), !,
                     calcula_diferencia(Conds,Normal).
  calcula_diferencia([Cond|Conds],[Cond|Normal]):-
                     calcula_diferencia(Conds,Normal).

predicates
  reduccion(string,reduc,real).
  reduce(atrib,atribs,condiciones,atrib).
  calc_sel(condicion,string,esquema,real).
  los_campos(atribs,listastring).
  depth_first(listalistastring,condiciones,listalistastring,
              condiciones).

clauses
  estima_selectividad(Campo,r(Cd1,Es1),r(Cd2,Es2),Selectividad):
                      reduccion(Campo,r(Cd1,Es1),Sel_inicial),
                      reduccion(Campo,r(Cd2,Es2),Sel_final),
                      Selectividad= Sel_final/Sel_inicial.

  reduccion(Campo,r(Cd,Es),Selec):-
            select2(Es,a5(E2,N2),Es2),
            select(c(Campo,_,_),E2),
            reduce(a5(E2,N2),Es2,Cd,a5(_,N3)),
            Selec= N3/N2, !.
  reduccion(_,_,1).

  reduce(a5(E1,N1),Rels,Conds,R2):-
        esquema_campos(E1,Ce1),
        select2(Conds,Cn,Conds2),
        condicion_campos(Cn,Cc),
        inter(Ce1,Cc,Cw1), Cw1=[Campo1],
        select2(Rels,a5(E3,N3),Rels2),
        esquema_campos(E3,Ce2),
        inter(Ce2,Cc,Cw2), Cw2=[Campo2],
        reduce(a5(E3,N3),Rels2,Conds2,a5(E4,_)),
        calc_sel(Cn,Campo2,E4,Selec),
        aplica_sel(Campo1,E1,N1,Selec,E5,N5),
        reduce(a5(E5,N5),Rels,Conds2,R2).
  reduce(Rel,_,_,Rel).

  calc_sel(cn(ig,_,_),Campo,Esq,Sel):- !,
           dom_campo(Campo,Dom),
           select(c(Campo,_,Val),Esq),
           Sel= Val/Dom.
  calc_sel(_,_,_,0.5).
```

```
es_aciclica(r(Cn,Es)):-
            findall(X,los_campos(Es,X),Cs),
            depth_first(Cs,Cn,_,[]).

los_campos(Atrs,Campos):- select(a5(Esq,_),Atrs).
                          esquema_campos(Esq,Campos).

depth_first([Vertice|Resto],Aristas,Resto2,Aristas2):-
            select2(Resto,V2,Ve2),
            union(Vertice,V2,Union),
            findall(X,condicion_subcon_campos(Aristas,Union,X).
                    Aris),
            not(Aris=[]), !,
            diferencia(Aristas,Aris,Ar2),
            depth_first([V2|Ve2],Ar2,Ve3,Ar3),
            depth_first([Vertice|Ve3],Ar3,Resto2,Aristas2).
depth_first([_|Resto],Aristas,Resto,Aristas).
```

```
/* eval.pro */

code= 2500

project "tesis2"
include "dominios.pro"


/* ============================================================ */
/* Predicados que manejan el preproceso para las relaciones */

/* ------------------------------------------------------------ */
/* Reglas para calculo de costo de operaciones de preproceso */
predicates
  costo_preproceso(nomoperadores,atrib,real).
  sel_costo(nomoperadores,esquema,real,real).
  costo_op(nomoperador,esquema,real,real).
  bloques(real,integer,real).

clauses
  costo_preproceso(Ops,a4(Esquema,Card,Size),Costo):-
                   bloques(Card,Size,B),
                   findall(X,sel_costo(Ops,Esquema,B,X),LC),
                   sum(LC,Costo).
  sel_costo(Ops,Esquema,Atrib,Costo):-
            select(Op,Ops), costo_op(Op,Esquema,Atrib,Costo).

  costo_op(scan(""),_,B,B):- !.
  costo_op(scan(C),E,B,Costo):- select(c(C,_,V),E),
max(V,B,Costo).
  costo_op(store,_,B,B).
  costo_op(ship(_),_,B,Costo):- factor_transmision(F), Costo=
F*B.
  costo_op(sort(_),_,B,Costo):- Costo= B*ln(B)+1.
  costo_op(index(_),_,B,Costo):- Costo= B*ln(B)+1.

  bloques(X1,X2,B):- bytes_por_bloque(X3), X4= X1*X2/X3,
ceil(X4,B).

/* ------------------------------------------------------------ */
/* Busqueda de un plan de ejecucion de acuerdo con objetivos */
predicates
  preproceso(nomoperadores,atrib,nomoperadores,atrib).
  preproceso2(nomoperador,atrib,nomoperadores).
  efecto_opers(nomoperadores,atrib,atrib).
  efecto_oper(nomoperador,atrib,atrib).

clauses
  /* Objetivos(in),Atributos(in),Programa(out),Costo(out) */
  programa_costea(Obj,a2(Esquema,_,_,Card,Size,Site,Sort,Indices,
                         Lugar,_),
                  Programa,Costo):-
        preproceso(Obj,a3(Site,Sort,Indices,Lugar),Programa,_),
        costo_preproceso(Programa,a4(Esquema,Card,Size),Costo).
```

```
  /* Objetivos(in),Atributos(in),Programa(out),Atributos(out) */
  preproceso([],Atr,[],Atr).
  preproceso([Obj|Objs],AtrIn,Pgm,AtrOut):-
            preproceso2(Obj,AtrIn,Pgm1),
            efecto_opers(Pgm1,AtrIn,Atr2),
            preproceso(Objs,Atr2,Pgm2,AtrOut),
            append(Pgm1,Pgm2,Pgm).

  preproceso2(scan(""),a3(_,_,_,secuencia),[]):- !.
  preproceso2(scan(C),a3(_,C,_,secuencia),[]):- !.
  preproceso2(scan(C),a3(S,O,I,secuencia),Pgm2):-
            preproceso2(sort(C),a3(S,O,I,secuencia),Pgm1),
            putlast(scan(""),Pgm1,Pgm2), !.
  preproceso2(scan(""),a3(_,_,_,disco),[scan("")]):- !.
  preproceso2(scan(C),a3(_,C,_,disco),[scan("")]):- !.
  preproceso2(scan(C),a3(_,_,I,disco),[scan(C)]):-
            select(C,I), !.
  preproceso2(scan(C),a3(S,O,I,disco),Pgm2):-
            preproceso2(sort(C),a3(S,O,I,disco),Pgm1),
            putlast(scan(""),Pgm1,Pgm2).

  preproceso2(ship(S),a3(S,_,_,_),[]):- !.
  preproceso2(ship(S),a3(T,O,I,L),Pgm2):-
            preproceso2(scan(""),a3(T,O,I,L),Pgm1),
            putlast(ship(S),Pgm1,Pgm2).

  preproceso2(store,a3(_,_,_,disco),[]):- !.
  preproceso2(store,_,[store]).

  preproceso2(sort(C),a3(_,C,_,_),[]):- !.
  preproceso2(sort(C),a3(S,O,I,L),Pgm2):-
            preproceso2(store,a3(S,O,I,L),Pgm1),
            putlast(sort(C),Pgm1,Pgm2).

  preproceso2(index(C),a3(_,_,I,_),[]):- select(C,I), !.
  preproceso2(index(C),a3(S,O,I,L),Pgm2):-
            preproceso2(store,a3(S,O,I,L),Pgm1),
            putlast(index(C),Pgm1,Pgm2).

  efecto_opers([],A,A).
  efecto_opers([Op|Ops],A1,A2):-
             efecto_oper(Op,A1,A3),
             efecto_opers(Ops,A3,A2).

  efecto_oper(scan(""),a3(S,O,_,_),a3(S,O,[],secuencia)):- !.
  efecto_oper(scan(C),a3(S,_,_,_),a3(S,C,[],secuencia)).
  efecto_oper(store,a3(S,O,_,_),a3(S,O,[],disco)).
  efecto_oper(ship(S),a3(_,O,_,_),a3(S,O,[],disco)).
  efecto_oper(index(C),a3(S,O,I,L),a3(S,O,[C|I],L)).
  efecto_oper(sort(C),a3(S,_,_,L),a3(S,C,[],L)).

/* ------------------------------------------------------------ */
/* Eleccion de una condicion par una seleccion dada             */
predicates
  elige_condicion(esquema,listastring,condiciones,condicion).
```

```
  elige_mejor(esquema,condicion,condiciones,condicion).
  es_mejor(esquema,condicion,condicion).

clauses
  elige_condicion(E,Cc,Cs,C):-
                findall(X,condicion_inters_campos(Cs,Cc,X),C1),
                C1=[C1|Cr],
                elige_mejor(E,C1,Cr,C), !.
  elige_condicion(E,_,[C1|Cr],C):- elige_mejor(E,C1,Cr,C).

  condicion_subcon_campos(Condiciones,Campos,Condicion):-
                          select(Condicion,Condiciones),
                          condicion_campos(Condicion,Cs),
                          subconjunto(Cs,Campos).
  condicion_inters_campos(Condiciones,Campos,Condicion):-
                          select(Condicion,Condiciones),
                          condicion_campos(Condicion,Cs),
                          not(inter(Cs,Campos,[])).

  elige_mejor(_,C,[],C):- !.
  elige_mejor(E,C1,[C2|Cr],C3):- es_mejor(E,C1,C2), !,
                                 elige_mejor(E,C1,Cr,C3).
  elige_mejor(E,C1,[C2|Cr],C3):- es_mejor(E,C2,C1), !,
                                 elige_mejor(E,C2,Cr,C3).
  elige_mejor(E,C1,[_|Cr],C3):- elige_mejor(E,C1,Cr,C3).

  es_mejor(_,cn(ig,_,_),cn(Op,_,_)):- not(Op=ig).
  es_mejor(_,cn(_,_,c(_)),cn(_,_,v(_))).
  es_mejor(E,cn(_,v(C1),_),cn(_,v(C2),_)):-
          select(c(C1,_,V1),E), select(c(C2,_,V2),E), V1<V2.


/*
=============================================================
*/
/* Estimacion de atributos de una expresion
*/

/* Estimacion del esquema y cualidades */
predicates
  ev0_1(esquema,listastring,campo).

clauses
  base0(vacia,a0([],[])).
  base0(rel(p(_,E,C,_,_,_,_,_)),a0(E,C)).
  ev0(o(sl,_,s(Cond)),[a0(E,C)],a0(E,C2)):- append(C,Cond,C2).
  ev0(o(pj,_,p(Campos)),[a0(E,C)],a0(E2,C2)):-
     findall(X,ev0_1(E,Campos,X),E2),
     esquema_campos(E2,Cs),
     findall(X,condicion_subcon_campos(C,Cs,X),C2).
  ev0(o(pd,_,_),[a0(E1,C1),a0(E2,C2)],a0(E3,C3)):-
     append(E1,E2,E3),
     append(C1,C2,C3).
  ev0(o(jn,_,s(Cond)),[a0(E1,C1),a0(E2,C2)],
```

```
      a0(E3,C3)):-
      append(E1,E2,E3),
      append(C1,C2,Cc),
      append(Cc,Cond,C3).
  ev0(o(sj,_,_),[a0(E1,C1),_],a0(E1,C1)).
  ev0(o(un,_,u(Remp1,Remp2)),[a0(E1,C1),a0(_,C2)],a0(E3,C3)):-
      remplaza_esquema(Remp1,E1,E3),
      remplaza_condiciones(Remp1,C1,Cc1),
      remplaza_condiciones(Remp2,C2,Cc2),
      inter(Cc1,Cc2,C3).

  ev0_1(Es,Cs,c(C,X1,X2)):- select(c(C,X1,X2),Es), select(C,Cs).

/* Estimacion del esquema, cualidad, reduccion, cardinalidad y
tamano */
predicates
  ev1_2(esquema,esquema,esquema,remplazos,remplazos,campo).
  ev1_3(esquema,real,reduc,esquema,real,reduc,condiciones,real,
        esquema,esquema,real).
  selecciones(condiciones,esquema,real,esquema,real).
  seleccion(condicion,esquema,real,esquema,real).
  img(esquema,real,string,campo).
  img_1(real,real,real).

clauses
  base1(vacia,a1([],[],r([],[]),0,0)).
  base1(rel(p(_,E,C,N,S,_,_,_)),a1(E,C,r([],[a5(E,N)]),N,S)).
  ev1(o(sl,Met,s(Cs)),[a1(E1,C1,_,N1,S1)],
      a1(E2,C2,r([],[a5(E2,N2)]),N2,S1)):-
      ev0(o(sl,Met,s(Cs)),[a0(E1,C1)],W), W=a0(_,C2),
      selecciones(Cs,E1,N1,E2,N2).
  ev1(o(pj,Met,p(Cs)),[a1(E1,C1,R1,N1,_)],
      a1(E2,C2,R1,N2,S2)):-
      ev0(o(pj,Met,p(Cs)),[a0(E1,C1)],W), W=a0(E2,C2),
      findall(X,select(c(_,X,_),E2),L1),
      sum(L1,S2),
      findall(X,select(c(_,_,X),E2),L2),
      times(L2,N3),
      min(N1,N3,N2).
  ev1(o(pd,Met,nulo),[a1(E1,C1,R1,N1,S1),a1(E2,C2,R2,N2,S2)],
      a1(E3,C3,R3,N3,S3)):-
      une_reduc(R1,R2,R3),
      ev0(o(pd,Met,nulo),[a0(E1,C1),a0(E2,C2)],W), W=a0(E3,C3),
      N3= N1*N2, S3= S1+S2.
  ev1(o(jn,Met,s(Conds)),[a1(E1,C1,R1,N1,S1),a1(E2,C2,R2,N2,S2)],
      a1(E3,C3,R3,N3,S3)):-
      une_reduc(R1,R2,Rk),
      une_reduc(r(Conds,[]),Rk,R3),
      ev0(o(jn,Met,s(Conds)),[a0(E1,C1),a0(E2,C2)],W), W=
a0(_,C3),
      esquema_campos(E1,Ce1),
      esquema_campos(E2,Ce2),
      select(Co,Conds),
      condicion_campos(Co,Cc),
      inter(Cc,Ce1,W1), W1= [Campo1],
```

```
    inter(Cc,Ce2,W2), W2= [Campo2],
    estima_selectividad(Campo1,R1,R3,Sel1),
    aplica_sel(Campo1,E1,N1,Sel1,Ef1,Nf1),
    estima_selectividad(Campo2,R2,R3,Sel2),
    aplica_sel(Campo2,E2,N2,Sel2,Ef2,Nf2),
    append(Ef1,Ef2,E3),
    select(c(Campo1,_,Val1),Ef1),
    N0= Nf1*Nf2/Val1, ceil(N0,N3),
    S3= S1+S2, !.
ev1(o(sj,Met,j(Cond)),[a1(E1,C1,R1,N1,S1),a1(E2,C2,R2,_,_)],
     a1(E3,C3,R3,N3,S1)):-
    une_reduc(R1,R2,Rk),
    une_reduc(r([Cond],[]),Rk,R3),
    ev0(o(sj,Met,j(Cond)),[a0(E1,C1),a0(E2,C2)],W), W=a0(E4,C3),
    esquema_campos(E4,Ce),
    condicion_campos(Cond,Cc),
    inter(Ce,Cc,W1), W1= [Campo],
    estima_selectividad(Campo,R1,R3,Sel),
    aplica_sel(Campo,E4,N1,Sel,E3,N3), !.
ev1(o(un,Met,u(Remp1,Remp2)),[a1(E1,C1,_,N1,S1),
                              a1(E2,C2,_,N2,_)],
     a1(E3,C3,r([],[a5(E3,N3)]),N3,S1)):-
    ev0(o(un,Met,u(Remp1,Remp2)),[a0(E1,C1),a0(E2,C2)],W),
       W=a0(E4,C3),
    invierte_remplazos(Remp1,Ri1),
    invierte_remplazos(Remp2,Ri2),
    findall(X,ev1_2(E4,E1,E2,Ri1,Ri2,X),E3),
    N3= N1+N2.

ev1_2(E,E1,E2,Remp1,Remp2,c(C,S,V)):-
      select(c(C,S,_),E),
      remplaza_campo(Remp1,C,C1), select(c(C1,_,V1),E1),
      remplaza_campo(Remp2,C,C2), select(c(C2,_,V2),E2),
      V= V1+V2.

ev1_3(Ei1,_,_,Ei2,_,_,[],Fac,Ei1,Ei2,Fac).
ev1_3(Ei1,Ni1,Ri1,Ei2,Ni2,Ri2,[Co|Cs],Fac,Ef1,Ef2,Fac2):-
      esquema_campos(Ei1,Cas1),
      esquema_campos(Ei2,Cas2),
      condicion_campos(Co,Cc),
      inter(Cas1,Cc,W1), W1= [Ca1],
      inter(Cas2,Cc,W2), W2= [Ca2],
      une_reduc(r([Co],[]),Ri1,Ra1),
      une_reduc(r([Co],[]),Ri2,Ra2),
      estima_selectividad(Ca1,Ri1,Ra1,Sel1),
      estima_selectividad(Ca2,Ri2,Ra2,Sel2),
      aplica_sel(Ca1,Ei1,Ni1,Sel1,Ea1,Nf1),
      aplica_sel(Ca2,Ei2,Ni2,Sel2,Ea2,Nf2),
      select(c(Ca2,_,Val2),Ei2), !,
      Fa= Fac*Sel1/Val2,
      Ev1_3(Ea1,Nf1,Ra1,Ea2,Nf2,Ra2,Cs,Fa,Ef1,Ef2,Fac2).
ev1_3(Ei1,Ni1,Ri1,Ei2,Ni2,Ri2,[Co|Cs],Fac,Ef1,Ef2,Fac2):-
      seleccion(Co,Ei1,Ni1,Ea1,Nf1), !,
      Sel1= Nf1/Ni1,
      Fa= Fac*Sel1,
```

```
      Ev1_3(Ea1,Nf1,Ri1,Ei2,Ni2,Ri2,Cs,Fa,Ef1,Ef2,Fac2).
  ev1_3(Ei1,Ni1,Ri1,Ei2,Ni2,Ri2,[Co|Cs],Fac,Ef1,Ef2,Fac2):-
      seleccion(Co,Ei2,Ni2,Ea2,Nf2),
      Sel2= Nf2/Ni2,
      Fa= Fac*Sel2,
      Ev1_3(Ei1,Ni1,Ri1,Ea2,Nf2,Ri2,Cs,Fa,Ef1,Ef2,Fac2).

  selecciones([],E,N,E,N).
  selecciones([C|Cs],E1,N1,E2,N2):-
            seleccion(C,E1,N1,E3,N3),
            selecciones(Cs,E3,N3,E2,N2).

  /* calcula cardinalidad y esquema a partir de una condicion */
  seleccion(cn(ig,v(C),c(_)),E1,N1,
            [c(C,S,1)|E2],N2):-
          select(c(C,S,V),E1), !,
          CC= N1/V, ceil(CC,N2),
          findall(X,img(E1,N2,C,X),E2).
  seleccion(cn(_,v(C),c(_)),E1,N1,
            [c(C,S,V2)|E2],N2):-
          select(c(C,S,V),E1), !,
          CC= N1/2, ceil(CC,N2), V2= V/2,
          findall(X,img(E1,N2,C,X),E2).
  seleccion(cn(ig,v(C1),v(C2)),E1,N1,
            E2,N2):- !,
          select(c(C1,_,V1),E1), M1= N1/V1,
          select(c(C2,_,V2),E1), M2= N1/V2, !,
          min(M1,M2,M3), ceil(M3,N2),
          findall(X,img(E1,N2,"?",X),E2).
  seleccion(cn(_,v(_),v(_)),E1,N1,E2,N2):-
          CC= N1/2, ceil(CC,N2),
          findall(X,img(E1,N2,"?",X),E2).

  /* calcula nuevo val. de acuerdo con la card. de la relacion
reducida */
  img(E1,N,C,c(C2,S2,V2)):- select(c(C2,S2,V1),E1), not(C=C2),
                            img_1(V1,N,V2).

  img_1(Val1,Card,Val1):- 2*Val1<Card, !.
  img_1(Val1,Card,Card):- Card<Val1/2, !.
  img_1(Val1,Card,Val2):- V2= (Val1+Card)/3, ceil(V2,Val2).

  aplica_sel(Campo,Esq,N,Sel,[c(Campo,Size,Val2)|Esq2],N2):-
            select(c(Campo,Size,Val),Esq), !,
            Val0= Sel*Val, ceil(Val0,Val2),
            N0= Sel*N, ceil(N0,N2),
            findall(X,img(Esq,N2,Campo,X),Esq2).



/* Determinacion del programa de ejecucion de un operador */
predicates
  determina_destino(integer,integer,integer).
  pgm_1(string,listastring,string,string).
  pgm_2(string,string,esquema,string,string).
```

```
  pgm_3(esquema,string,string,condiciones,condicion).

clauses
  /* estimacion de parametros para metodos de seleccion */
  pgm(o(sl,m(simple,N),Cs),[a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1)],
      a2(E2,C2,R2,N2,S2,P1,O1,[],secuencia,Costo2),
      [Pgm],[]):-
     ev1(o(sl,m(simple,N),Cs),[a1(E1,C1,R1,N1,S1)],W),
        W=a1(E2,C2,R2,N2,S2),
     programa_costea([scan("")],
                     a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),Pgm,Costo2).
  pgm(o(sl,m(indice,N),s(Cs)),
      [a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1)],
      a2(E2,C2,R2,N2,S2,P1,Cp,[],secuencia,Costo2),
      [Pgm],[cn(Cmp,v(Cp),E12)]):-
     ev1(o(sl,m(indice,N),s(Cs)),[a1(E1,C1,R1,N1,S1)],W),
        W=a1(E2,C2,R2,N2,S2),
     elige_condicion(E1,I1,Cs,cn(Cmp,v(Cp),E12)),
     programa_costea([index(Cp)],
                     a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),Pgm,X1),
     select(c(Cp,_,X2),E2), !, Costo2= X1+X2.

  /* estimacion de parametros para metodos de proyeccion */
  pgm(o(pj,m(simple,N),p(Cs)),
      [a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1)],
      a2(E2,C2,R2,N2,S2,P1,O2,[],disco,Costo2),
      [Pgm1,Pgm2],[]):-
     ev1(o(pj,m(simple,N),p(Cs)),[a1(E1,C1,R1,N1,S1)],W),
        W=a1(E2,C2,R2,N2,S2),
     pgm_1(O1,Cs,O2,O3),
     programa_costea([scan("")],
                     a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),Pgm1,X1),
     programa_costea([sort(O3)],
                     a2(E2,C2,R2,N2,S2,P1,O2,[],secuencia,0),
                     Pgm2,X2),
     Costo2= X1+X2.

  /* estimacion de parametros para metodos de producto */
  pgm(o(pd,m(loop,N),nulo),[a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),
                            a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2)],
      a2(E3,C3,R3,N3,S3,P3,O1,[],secuencia,Costo3),
      [Pgm1,Pgm2],[]):-
     ev1(o(pd,m(loop,N),nulo),[a1(E1,C1,R1,N1,S1),
                               a1(E2,C2,R2,N2,S2)],
         W), W=a1(E3,C3,R3,N3,S3),
     determina_destino(N,P2,P3),
     programa_costea([ship(P3),scan("")],
                     a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),Pgm1,X1),
     programa_costea([ship(P3),store],
                     a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2),Pgm2,X2),
     bloques(N1,S1,B1),
     bloques(N2,S2,B2),
     Costo3= X1+X2+B1*B2.

  /* estimacion de parametros para metodos de join */
```

```
pgm(o(jn,m(loop,N),s(Cs)),[a2(E1,C1,R1,N1,S1,P1,O1,I1.L1,Ct1),
                           a2(E2,C2,R2,N2,S2,P2,O2.I2,L2,Ct2)],
    a2(E3,C3,R3,N3,S3,P3,O1,[],secuencia,Costo3).
    [Pgm1,Pgm2],[]):-
   ev1(o(jn,m(loop,N),s(Cs)),[a1(E1,C1,R1.N1,S1).
                          a1(E2,C2.R2,N2,S2)],
       W), W=a1(E3,C3,R3,N3,S3),
   determina_destino(N,P2,P3),
   programa_costea([ship(P3),scan("")],
                   a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),Pgm1.X1),
   programa_costea([ship(P3),store],
                   a2(E2,C2,R2,N2,S2,P2,O2,I2.L2,Ct2),Pgm2,X2),
   bloques(N1,S1,B1),
   bloques(N2,S2,B2),
   Costo3= X1+X2+B1*B2.
pgm(o(jn,m(indice,N),s(Cs)),
    [a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),
     a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2)],
    a2(E3,C3.R3,N3,S3,P3,O1,[],secuencia,Costo3),
    [Pgm1,Pgm2],[cn(Cmp,v(Cp1),v(Cp2))]):-
   ev1(o(jn,m(indice,N),s(Cs)),[a1(E1,C1,R1,N1,S1),
                                a1(E2,C2,R2,N2,S2)],
       W), W=a1(E3,C3,R3,N3,S3),
   append(E1,E2,E4),
   elige_condicion(E4,I2,Cs,cn(Cmp,v(Cp1),v(Cp2))),
   pgm_2(Cp1,Cp2,E1,_,Cp),
   determina_destino(N,P2,P3),
   programa_costea([ship(P3),scan("")],
                   a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),Pgm1,X1),
   programa_costea([ship(P3),index(Cp)],
                   a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2),Pgm2,X2),
   Costo3= X1+X2+N1.
pgm(o(jn,m(merge,N),s(Cs)),[a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),
                            a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2)],
    a2(E3,C3,R3.N3,S3,P3,O1,[],secuencia,Costo3),
    [Pgm1,Pgm2],[cn(Cmp,v(Cp1),v(Cp2))]):-
   ev1(o(jn,m(merge,N),s(Cs)),[a1(E1,C1,R1,N1,S1),
                               a1(E2,C2,R2,N2,S2)],
       W), W=a1(E3,C3,R3,N3,S3),
   append(E1,E2,E4),
   pgm_3(E4,O1,O2,Cs,cn(Cmp,v(Cp1),v(Cp2))),
   pgm_2(Cp1,Cp2,E1,C1p,C2p),
   determina_destino(N,P2,P3),
   programa_costea([ship(P3),scan(C1p)],
                   a2(E1,C1,R1,N1,S1.P1,O1,I1,L1,Ct1),Pgm1,X1),
   programa_costea([ship(P3),scan(C2p)],
                   a2(E2,C2,R2,N2,S2,P2,O2,I2.L2,Ct2),Pgm2,X2),
   Costo3= X1+X2.

/* estimacion de parametros para metodos de semijoin */
pgm(o(sj,m(loop,N),Cond),[a2(E1,C1.R1,N1,S1,P1,O1,I1,L1,Ct1),
                          a2(E2.C2,R2,N2,S2,P2.O2,I2,L2,Ct2)],
    a2(E3,C3,R3,N3,S3,P1,O1,[],secuencia,Costo3),
    [Pgm1,Pgm2],[]):-
   ev1(o(sj,m(loop.N),Cond),[a1(E1,C1,R1,N1,S1),
```

```
                                    a1(E2,C2,R2,N2,S2)],
        W), W=a1(E3,C3,R3,N3,S3),
    programa_costea([store],
                    a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),Pgm1,X1),
    programa_costea([ship(P1),scan("")],
                    a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2),Pgm2,X2),
    bloques(N1,S1,B1),
    bloques(N2,S2,B2),
    Costo3= X1+X2+B1*B2.
pgm(o(sj,m(indice,N),j(Cs)),
     [a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),
      a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2)],
     a2(E3,C3,R3,N3,S3,P1,O1,[],secuencia,Costo3),
     [Pgm1,Pgm2],[cn(Cmp,v(Cp1),v(Cp2))]):-
    ev1(o(sj,m(indice,N),j(Cs)),[a1(E1,C1,R1,N1,S1),
                                 a1(E2,C2,R2,N2,S2)],
        W), W=a1(E3,C3,R3,N3,S3),
    Cs=cn(Cmp,v(Cp1),v(Cp2)),
    pgm_2(Cp1,Cp2,E1,_,Cp),
    programa_costea([index(Cp)],
                    a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),Pgm1,X1),
    programa_costea([ship(P1),scan("")],
                    a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2),Pgm2,X2),
    Costo3= X1+X2+N1.
pgm(o(sj,m(merge,N),j(Cs)),[a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),
                            a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2)],
     a2(E3,C3,R3,N3,S3,P1,O1,[],secuencia,Costo3),
     [Pgm1,Pgm2],[cn(Cmp,v(Cp1),v(Cp2))]):-
    ev1(o(sj,m(merge,N),j(Cs)),[a1(E1,C1,R1,N1,S1),
                                a1(E2,C2,R2,N2,S2)],
        W), W=a1(E3,C3,R3,N3,S3),
    Cs=cn(Cmp,v(Cp1),v(Cp2)),
    pgm_2(Cp1,Cp2,E1,C1p,C2p),
    programa_costea([scan(C1p)],
                    a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),Pgm1,X1),
    programa_costea([ship(P1),scan(C2p)],
                    a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2),Pgm2,X2),
    Costo3= X1+X2.

/* estimacion de parametros para metodos de union */
pgm(o(un,m(simple,N),u(Remp1,Remp2)),
     [a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),
      a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2)],
     a2(E3,C3,R3,N3,S3,P3,O3,[],disco,Costo3),
     [Pgm1,Pgm2,Pgm3],[]):-
    ev1(o(un,m(simple,N),u(Remp1,Remp2)),[a1(E1,C1,R1,N1,S1),
                                          a1(E2,C2,R2,N2,S2)],
        W), W=a1(E3,C3,R3,N3,S3),
    determina_destino(N,P2,P3),
    programa_costea([ship(P3),scan("")],
                    a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Ct1),Pgm1,X1),
    programa_costea([ship(P3),scan("")],
                    a2(E2,C2,R2,N2,S2,P2,O2,I2,L2,Ct2),Pgm2,X2),
    remplaza_campo(Remp1,O1,O4), esquema_campos(E3,L),
    pgm_1(O4,L,_,O3),
```

```
      programa_costea([store,sort(O3)],
                      a2(E3,C3,R3,N3,S3,P3,"?",[],secuencia,0),
                      Pgm3,X3),
      Costo3= X1+X2+X3.

  pgm_1(Campo,Campos,Campo,Campo):- select(Campo,Campos), !.
  pgm_1(_,Campos,"?",Campo):- select(Campo,Campos), !.

  determina_destino(0,N,N):- !.
  determina_destino(N,_,N).

  pgm_2(Campo1,Campo2,Esquema,Campo1,Campo2):-
       select(c(Campo1,_,_),Esquema), !.
  pgm_2(Campo1,Campo2,_,Campo2,Campo1).

  pgm_3(_,C1,C2,Cs,cn(ig,E1,E2)):-
       findall(X,condicion_subcon_campos(Cs,[C1,C2],X),L),
       select(cn(ig,E1,E2),L), !.
  pgm_3(E,C1,C2,Cs,cn(ig,E1,E2)):-
       elige_condicion(E,[C1,C2],Cs,cn(ig,E1,E2)).

/* Estimacion del esquema, cardinalidad, tamano y costo */
clauses
  base2(vacia,a2([],[],r([],[]),0,0,1,"?",[],disco,0)).
  base2(rel(p(_,Esquema,Cualidad,Card,Size,[Site|_],
              Sort,Indices)),
        a2(Esquema,Cualidad,r([],[a5(Esquema,Card)]),
           Card,Size,Site,Sort,Indices,disco,0)).

  ev2(Op,Atrs,Atr):- pgm(Op,Atrs,Atr,_,_).


/* Aplica una formula de calculo en toda la expresion */
clauses
  eval(Regla,_,vacia,A):- call0(Regla,vacia,A).
  eval(Regla,_,rel(R),A):- call0(Regla,rel(R),A).
  eval(Regla0,Regla,exp(Op,[E1]),A):-
      eval(Regla0,Regla,E1,A1),
      call1(Regla,Op,[A1],A).
  eval(Regla0,Regla,exp(Op,[E1,E2]),A):-
      eval(Regla0,Regla,E1,A1),
      eval(Regla0,Regla,E2,A2),
      call1(Regla,Op,[A1,A2],A).

  call0(base0,E,A):- base0(E,A).
  call0(base1,E,A):- base1(E,A).
  call0(base2,E,A):- base2(E,A).
  call1(ev0,Op,As,A):- ev0(Op,As,A).
  call1(ev1,Op,As,A):- ev1(Op,As,A).
  call1(ev2,Op,As,A):- ev2(Op,As,A).
```

```
/* subexp.pro */

code= 2000

project "tesis2"
include "dominios.pro"


/* ================================================== */
/* Predicados para el manejo de subexpresiones comunes */

predicates
  exp_subexp(expresion,subexpresion).
  empaca2(subexpresion,symbol).
  subexp_exp(subexpresion,expresion).
  call2(string,string,subexpresion,atrib).
  lista_atrib(listasymbol,atrib).
  depth_first(symbol,listasymbol,listasymbol,real).
  visita(listasymbol,listasymbol,listasymbol,real).

clauses
  empaca(Exp,Ident):- exp_subexp(Exp,Subexp),
                      empaca2(Subexp,Ident).

  exp_subexp(vacia,vacia).
  exp_subexp(rel(Profile),rel(Profile)).
  exp_subexp(exp(Op,[]),exp(Op,[])).
  exp_subexp(exp(Op,[E|Es]),exp(Op,[I|Is])):-
           empaca(E,I), exp_subexp(exp(Op,Es),exp(_,Is)).

/*  empaca2(Subexp,_):- write(Subexp,' '), fail.*/
  empaca2(Subexp,Ident):-
        mesh(Ident,_,Subexp)/*, write("ya estaba\n")*/, !;
        gensym(e,Ident), call2(base2,ev2,Subexp,Atr),
        asserta(mesh(Ident,Atr,Subexp)).

  desempaca(Ident,Exp):- mesh(Ident,_,Subexp), !,
subexp_exp(Subexp,Exp).

  subexp_exp(vacia,vacia).
  subexp_exp(rel(Profile),rel(Profile)).
  subexp_exp(exp(Op,[]),exp(Op,[])).
  subexp_exp(exp(Op,[I|Is]),exp(Op,[E|Es])):-
           desempaca(I,E), subexp_exp(exp(Op,Is),exp(_,Es)).

  call2(Regla,_,rel(R),A)      :- call0(Regla,rel(R),A).
  call2(_,Regla,exp(Op,Ids),A):- findall(X,lista_atrib(Ids,X),L),
                                 call1(Regla,Op,L,A).
  lista_atrib(Ids,X):- select(I,Ids), mesh(I,X,_).

  destruye_mesh:- reset_gensym(e), fail.
  destruye_mesh:- retract(mesh(_,_,_)), fail.
  destruye_mesh.

  exp_costo(Exp,Costo):- empaca(Exp,Id), id_costo(Id,Costo),
```

```
destruye_mesh.

  id_costo(Id,Costo):- depth_first(Id,[],_,C1),
                       mesh(Id,Atr,_), !, sitio_usuario(S),
                       programa_costea([ship(S)],Atr,_,C2),
                       Costo= C1+C2.
  depth_first(Id,Visi,Visi,0)      :- select(Id,Visi), !.
  depth_first(Id,Visi,[Id|Visi],C):-
mesh(Id,a2(_,_,_,_,_,_,_,_,_,C),vacia),
                                      !.
  depth_first(Id,Visi,[Id|Visi],C):-
mesh(Id,a2(_,_,_,_,_,_,_,_,_,C),rel(_)),
                                      !.
  depth_first(Id,Visi1,Visi2,C)    :-
            mesh(Id,a2(_,_,_,_,_,_,_,_,_,C1),
            exp(_,Ids)), !,
            visita([Id|Visi1],Ids,Visi2,C2),
            C= C1+C2.
  visita(Visi,[],Visi,0).
  visita(Visi1,[Id|Ids],Visi2,C):-
        depth_first(Id,Visi1,Visi3,C1),
        visita(Visi3,Ids,Visi2,C2),
        C= C1+C2.

predicates
  depth_firstp(symbol,listasymbol,listasymbol).
  visitap(listasymbol,listasymbol,listasymbol).
  imp_programa(integer,nomoperadores,symbol,symbol).
  imp_programa2(integer,nomoperadores,symbol,symbol).
  imp_opers(nomoperadores).

clauses
  programa_ejecucion(Id):- depth_firstp(Id,[],_),
                           mesh(Id,Atr,_), !,
                           sitio_usuario(S),
                           programa_costea([ship(S)],Atr,Pgm,Co),
                           write(Co), nl,
                           imp_programa2(S,Pgm,Id,res),
                           reset_gensym(i).
  depth_firstp(Id,Visi,Visi):- select(Id,Visi), !.
  depth_firstp(Id,Visi,[Id|Visi]):-
            mesh(Id,_,vacia), !,
            writef("%2: %4(%) <-- vacia\n",'-',Id,'-').
  depth_firstp(Id,Visi,[Id|Visi]):-
            mesh(Id,a2(Esq,_,Red,Card,_,S,_,Ix,_,Co),
                 rel(p(N,_,_,_,_,_,_,_))), !,
            descompone(N,N1,N2),
            writef("%2: %4(%) <-- % - %(%)\n",S,Id,Co,N1,N2,S),
            write(Card," ",Esq,'\n',Red,Ix,'\n').
  depth_firstp(Id,Visi1,Visi2):-
            mesh(Id,a2(Esq,_,Red,Card,_,S,_,Ix,_,Co),
                 exp(o(sl,m(M,N),s(Arg)),[I1])),
            visitap([Id|Visi1],[I1],Visi2),
            mesh(I1,A1,_), !,
            pgm(o(sl,m(M,N),s(Arg)),[A1],_,Pgm,Cs), Pgm=[Pgm1],
```

```
                imp_programa(S,Pgm1,I1,R1),
                writef("%2: %4(%) <-- sl ",S,Id,Co),
                write(M,"("), print3(Cs), write(") "), print3(Arg),
                writef(" : %\n",R1),
                write(Card," ",Esq,'\n',Red,Ix,'\n').
  depth_firstp(Id,Visi1,Visi2):-
                mesh(Id,a2(Esq,_,Red,Card,_,S,_,Ix,_,Co),
                     exp(o(pj,m(M,N),p(Arg)),[I1])),
                visitap([Id|Visi1],[I1],Visi2),
                mesh(I1,A1,_), !,
                pgm(o(pj,m(M,N),p(Arg)),[A1],_,Pgm,_),
                Pgm=[Pgm1,Pgm2],
                imp_programa(S,Pgm1,I1,R1),
                gensym(i,In),
                writef("%2: %4(%) <-- pj ",S,In,Co), write(M," "),
                print4(Arg),
                writef(" : %\n",R1),
                imp_programa2(S,Pgm2,In,Id),
                write(Card," ",Esq,'\n',Red,Ix,'\n').
  depth_firstp(Id,Visi1,Visi2):-
                mesh(Id,a2(Esq,_,Red,Card,_,S,_,Ix,_,Co),
                        exp(o(jn,m(M,N),s(Arg)),[I1,I2])),
                visitap([Id|Visi1],[I1,I2],Visi2),
                mesh(I1,A1,_), mesh(I2,A2,_), !,
                pgm(o(jn,m(M,N),s(Arg)),[A1,A2],_,Pgm,Cs),
Pgm=[Pgm1,Pgm2],
                imp_programa(S,Pgm1,I1,R1),
                imp_programa(S,Pgm2,I2,R2),
                writef("%2: %4(%) <-- jn ",S,Id,Co),
                write(M,"("), print3(Cs), write(") "), print3(Arg),
                writef(" : % %\n",R1,R2),
                write(Card," ",Esq,'\n',Red,Ix,'\n').
  depth_firstp(Id,Visi1,Visi2):-
                mesh(Id,a2(Esq,_,Red,Card,_,S,_,Ix,_,Co),
                        exp(o(sj,m(M,N),j(Arg)),[I1,I2])),
                visitap([Id|Visi1],[I1,I2],Visi2),
                mesh(I1,A1,_), mesh(I2,A2,_), !,
                pgm(o(sj,m(M,N),j(Arg)),[A1,A2],_,Pgm,_),
Pgm=[Pgm1,Pgm2],
                imp_programa(S,Pgm1,I1,R1),
                imp_programa(S,Pgm2,I2,R2),
                writef("%2: %4(%) <-- sj ",S,Id,Co),
                write(M,"("), print3_1(Arg), write(") "),
                writef(" : % %\n",R1,R2),
                write(Card," ",Esq,'\n',Red,Ix,'\n').
  depth_firstp(Id,Visi1,Visi2):-
                mesh(Id,a2(Esq,_,Red,Card,_,S,_,Ix,_,Co),
                     exp(o(pd,m(M,N),Arg),[I1,I2])),
                visitap([Id|Visi1],[I1,I2],Visi2),
                mesh(I1,A1,_), mesh(I2,A2,_), !,
                pgm(o(pd,m(M,N),Arg),[A1,A2],_,Pgm,_),
                Pgm=[Pgm1,Pgm2],
                imp_programa(S,Pgm1,I1,R1),
                imp_programa(S,Pgm2,I2,R2),
                writef("%2: %4(%) <-- pd ",S,Id,Co), write(M),
```

```
                writef(" : % %\n",R1,R2),
                write(Card," ",Esq,'\n',Red,Ix,'\n').
  depth_firstp(Id,Visi1,Visi2):-
                mesh(Id,a2(Esq,_,Red,Card,_,S,_,Ix,_,Co),
                     exp(o(un,m(M,N),Arg),[I1,I2])),
                visitap([Id|Visi1],[I1,I2],Visi2),
                mesh(I1,A1,_), mesh(I2,A2,_), !,
                pgm(o(un,m(M,N),Arg),[A1,A2],_,Pgm,_),
Pgm=[Pgm1,Pgm2,Pgm3],
                imp_programa(S,Pgm1,I1,R1),
                imp_programa(S,Pgm2,I2,R2),
                gensym(i,In),
                writef("%2: %4(%) <-- un ",S,In,Co), write(M),
                writef(" : % %\n",R1,R2),
                imp_programa2(S,Pgm3,In,Id),
                write(Card," ",Esq,'\n',Red,Ix,'\n').
  visitap(Visi,[],Visi).
  visitap(Visi1,[Id|Ids],Visi2):- depth_firstp(Id,Visi1,Visi3),
                                  visitap(Visi3,Ids,Visi2).
  imp_programa(_,[],R,R).
  imp_programa(S,Pgm,R1,R2):- gensym(i,R2),
imp_programa2(S,Pgm,R1,R2).
  imp_programa2(S,Pgm,R1,R2):- writef("%2: %4 <-- ",S,R2),
                               imp_opers(Pgm), writef("%\n",R1).
  imp_opers([]).
  imp_opers([Op|Ops]):- imp_opers(Ops), write(Op," < ").
```

```
/* reglas.pro */

code= 2500

project "tesis2"
include "dominios.pro"


/* ========================================================== */
/* Reglas de transformacion                                   */

clauses
  metodo(pj,m(simple,0)).
  metodo(sl,m(simple,0)).
  metodo(sl,m(indice,0)).
  metodo(pd,m(loop,0)).
  metodo(jn,m(loop,0)).
  metodo(jn,m(indice,0)).
  metodo(jn,m(merge,0)).
  metodo(un,m(simple,0)).
  metodo(sj,m(loop,0)).
  metodo(sj,m(indice,0)).
  metodo(sj,m(merge,0)).

  un_metodo(Nomop,Metop):- metodo(Nomop,Metop), !.

/* ---------------------------------------------------------- */
/* Reglas elemetales de transformacion                        */
predicates
  mejora(expresion,expresion).
  mejora1(expresion,expresion).
  mejora2(expresion,expresion).
  mejora3(expresion,expresion).
  mejora4(expresion,expresion).
  mejora5(expresion,expresion).
  mejora5_1(condiciones,atrib,atrib,condiciones,condiciones,
            condiciones).
  mejora5_2(condiciones,atrib,condiciones,condiciones).
  mejora6(expresion,expresion).
  mejora6_1(listastring,condiciones,expresion,expresion).
  mejora7(expresion,expresion).
/*  mejora8(expresion,expresion).*/
  aplica(string,string,expresion,expresion).
  call(string,expresion,expresion).

clauses
  /* mejora una expresion mediante el uso de alguna regla */
  mejora(E1,E2):- mejora1(E1,E2); mejora2(E1,E2); mejora3(E1,E2);
                  mejora4(E1,E2); mejora5(E1,E2); mejora6(E1,E2);
                  mejora7(E1,E2)/*; mejora8(E1,E2)*/.
  mejora(exp(Op,[E1]),exp(Op,[E2])):- mejora(E1,E2).
  mejora(exp(Op,[E1,E2]),exp(Op,[E3,E2])):- mejora(E1,E3).
  mejora(exp(Op,[E1,E2]),exp(Op,[E1,E3])):- mejora(E2,E3).

  /* conmutatividad de operadores unarios */
```

```
  mejora1(exp(o(sl,Met1,Arg1),[exp(o(pj,Met2,Arg2),Exs1)]),
          exp(o(pj,Met2,Arg2),[exp(o(sl,Met1,Arg1),Exs1)])).

  /* idempotencia de operadores unarios */
  mejora2(exp(o(sl,Met1,s(Arg1)),[exp(o(sl,_,s(Arg2)),Exs1)]),
          exp(o(sl,Met1,s(Arg3)),Exs1)):-
         append(Arg1,Arg2,Arg3), !.
  mejora2(exp(o(pj,Met1,Arg1),[exp(o(pj,_,_),Exs1)]),
          exp(o(pj,Met1,Arg1),Exs1)).

  /* operadores compuestos */
  mejora3(exp(o(sl,_,Arg1),[exp(o(pd,_,_),Exs1)]),
          exp(o(jn,Met1,Arg1),Exs1)):-
         un_metodo(jn,Met1).
  mejora3(exp(o(sl,_,s(Arg1)),[exp(o(jn,Met1,s(Arg2)),Exs1)]),
          exp(o(jn,Met1,s(Arg3)),Exs1)):-
         append(Arg1,Arg2,Arg3).

  /* eliminacion de operadores inutiles */
  mejora4(exp(o(jn,_,s([])),Exs1),
          exp(o(pd,Met1,nulo),Exs1)):- un_metodo(pd,Met1).
  mejora4(exp(o(sl,_,s([])),[E1]),E1).
  mejora4(exp(o(pj,_,p(Campos1)),[E1]),E1):-
         eval(base0,ev0,E1,Atr1), Atr1=a0(Esq1,_),
         esquema_campos(Esq1,Campos2),
         conjuntos_iguales(Campos1,Campos2).
  mejora4(exp(o(sj,_,_),[E1,E2]),E1):-
         eval(base2,ev2,E1,A1), A1=a2(_,_,_,_,_,S,_,_,_,_),
         eval(base2,ev2,E2,A2), A2=a2(_,_,_,_,_,S,_,_,_,_).

  /* distributividad de operadores unarios que mejora la consulta
*/
  mejora5(exp(o(jn,Met1,s(Arg1)),[E1,E2]),
          exp(o(jn,Met1,s(Arg2)),[E3,E4])):-
         un_metodo(sl,Met2),
         eval(base2,ev2,E1,Atr1),
         eval(base2,ev2,E2,Atr2),
         mejora5_1(Arg1,Atr1,Atr2,Arg2,Arg3,Arg4),
         not(Arg1=Arg2),
         aplica("?",mejora4,exp(o(sl,Met2,s(Arg3)),[E1]),E3),
         aplica("?",mejora4,exp(o(sl,Met2,s(Arg4)),[E2]),E4).
  mejora5(exp(o(sl,Met1,s(Arg1)),
          [exp(o(un,Met2,u(Remp1,Remp2)),[E1,E2])]),
          exp(o(un,Met2,u(Remp1,Remp2)),
              [exp(o(sl,Met1,s(Arg2)),[E1]),exp(o(sl,Met1,s(Arg3)
),[E2])])):-
         invierte_remplazos(Remp1,Imp1),
         remplaza_condiciones(Imp1,Arg1,Arg2),
         invierte_remplazos(Remp2,Imp2),
         remplaza_condiciones(Imp2,Arg1,Arg3).
  mejora5(exp(o(pj,Met1,p(Arg1)),[exp(o(un,Met2,u(Remp1,Remp2)),
          [E1,E2])]),
          exp(o(un,Met2,u(Remp1,Remp2)),
              [exp(o(pj,Met1,p(Arg2)),[E1]),exp(o(pj,Met1,p(Arg3)
),[E2])])):-
```

```
            invierte_remplazos(Remp1,Imp1),
            remplaza_campos(Imp1,Arg1,Arg2),
            invierte_remplazos(Remp2,Imp2),
            remplaza_campos(Imp2,Arg1,Arg3).

   mejora5_1(Arg1,a2(E1,_,_,_,_,S1,_,_,_,_),
              a2(E2,_,_,_,_,S2,_,_,_,_),
              Arg2,Arg3,Arg4):-
             not(S1=S2), !,
             esquema_campos(E1,Camp1),
             esquema_campos(E2,Camp2),
             divseleccion(Arg1,Camp1,Arg3,Arg5),
             divseleccion(Arg5,Camp2,Arg4,Arg2).
   mejora5_1(Arg1,A1,A2,Arg2,Arg3,Arg4):-
             mejora5_2(Arg1,A1,Arg3,Arg5),
             mejora5_2(Arg5,A2,Arg4,Arg2).
   mejora5_2(Arg1,a2(E1,_,_,_,_,_,_,[],_,_),Arg2,Arg3):- !,
             esquema_campos(E1,C1),
             divseleccion(Arg1,C1,Arg2,Arg3).
   mejora5_2(Arg1,_,[],Arg1).

   /* normalizacion de condiciones y deteccion de contradicciones
 */
   mejora6(exp(o(sl,Met,Arg1),[E1]),E2):-
           eval(base0,ev0,E1,Atr1), Atr1=a0(Esq1,Cual1),
           esquema_campos(Esq1,L),
           mejora6_1(L,Cual1,exp(o(sl,Met,Arg1),[E1]),E2).
   mejora6(exp(o(jn,Met,Arg1),[E1,E2]),E3):-
           eval(base0,ev0,E1,Atr1), Atr1=a0(Esq1,Cual1),
           eval(base0,ev0,E2,Atr2), Atr2=a0(Esq2,Cual2),
           esquema_campos(Esq1,L1),
           esquema_campos(Esq2,L2),
           append(L1,L2,L), append(Cual1,Cual2,Cual),
           mejora6_1(L,Cual,exp(o(jn,Met,Arg1),[E1,E2]),E3).
   mejora6_1(L,Cual,exp(o(sl,Met,s(Arg1)),[E1]),
              exp(o(sl,Met,s(Arg2)),[E1])):-
             normaliza(L,Cual,Arg1,Arg2), !, not(Arg1=Arg2).
   mejora6_1(L,Cual,exp(o(jn,Met,s(Arg1)),[E1,E2]),
              exp(o(jn,Met,s(Arg2)),[E1,E2])):-
             normaliza(L,Cual,Arg1,Arg2), !, not(Arg1=Arg2).
   mejora6_1(_,_,_,vacia).

   /* simplificacion de expresiones con la relacion vacia */
   mejora7(exp(o(sl,_,_),[vacia]),vacia).
   mejora7(exp(o(pj,_,_),[vacia]),vacia).
   mejora7(exp(o(pd,_,_),Es),vacia):- select(vacia,Es), !.
   mejora7(exp(o(jn,_,_),Es),vacia):- select(vacia,Es), !.
   mejora7(exp(o(un,_,u(Remp1,_)),[E1,vacia]),E2):-
           remplaza_expresion(Remp1,E1,E2), !.
   mejora7(exp(o(un,_,u(_,Remp1)),[vacia,E1]),E2):-
           remplaza_expresion(Remp1,E1,E2).

/*   /* Elimina una relacion en producto cartesiano */
   mejora8(exp(o(pj,_,p(Arg1)),[exp(o(pd,_,_),[E1,_])]),E1):-
           eval(base0,ev0,E1,A1), A1= a0(Es1,_),
```

```
        esquema_campos(Es1,Cam1),
        subconjunto(Arg1,Cam1).
  mejora8(exp(o(pj,_,p(Arg1)),[exp(o(pd,_,_),[_,E2])]),E2):-
        eval(base0,ev0,E2,A2), A2= a0(Es2,_),
        esquema_campos(Es2,Cam2),
        subconjunto(Arg1,Cam2).*/

  divseleccion(Condiciones,Campos,Pertenecen,Sobran):-
              findall(X,condicion_subcon_campos(Condiciones,
                                                Campos,X),
                      Pertenecen),
              diferencia(Condiciones,Pertenecen,Sobran).

predicates
  regla1(expresion,expresion).
  regla2(expresion,expresion).
  regla2_1(expresion,expresion).
  regla3(expresion,expresion).
  regla4(expresion,expresion).
  regla5(expresion,expresion).
  regla5_1(listastring,expresion,expresion).
  regla6(expresion,expresion).

clauses
  /* conmutatividad de operadores binarios */
  regla1(exp(o(pd,Met,Arg),[E1,E2]),exp(o(pd,Met,Arg),[E2,E1])).
  regla1(exp(o(jn,Met,Arg),[E1,E2]),exp(o(jn,Met,Arg),[E2,E1])).
  regla1(exp(o(un,Met,u(R1,R2)),[E1,E2]),exp(o(un,Met,u(R2,R1)),
        [E2,E1])).

  /* asociatividad de operadores binarios */
  regla2(exp(o(jn,Met1,s(Arg1)),
        [exp(o(jn,Met2,s(Arg2)),[E1,E2]),E3]),
        exp(o(jn,Met2,s(Arg3)),[E1,exp(o(jn,Met1,s(Arg4)),
        [E2,E3])])):-
       eval(base0,ev0,E2,Atr2), Atr2=a0(Esquema2,_),
       eval(base0,ev0,E3,Atr3), Atr3=a0(Esquema3,_),
       append(Esquema2,Esquema3,Esq),
       append(Arg1,Arg2,Arg),
       esquema_campos(Esq,Camp),
       divseleccion(Arg,Camp,Arg4,Arg3),
       not(Arg4=[]).
  regla2(exp(o(jn,Met1,s(Arg1)),
        [E1,exp(o(jn,Met2,s(Arg2)),[E2,E3])]),
        exp(o(jn,Met2,s(Arg3)),[exp(o(jn,Met1,s(Arg4)),
            [E1,E2]),E3])):-
       eval(base0,ev0,E1,Atr1), Atr1=a0(Esquema1,_),
       eval(base0,ev0,E2,Atr2), Atr2=a0(Esquema2,_),
       append(Esquema1,Esquema2,Esq),
       append(Arg1,Arg2,Arg),
       esquema_campos(Esq,Camp),
       divseleccion(Arg,Camp,Arg4,Arg3),
       not(Arg4=[]).
  regla2(E1,E2):- regla2_1(E1,E2).
  regla2(E1,E2):- regla2_1(E2,E1).
```

```
regla2_1(exp(o(pd,Met1,Arg1),
          [exp(o(pd,Met2,Arg2),[E1,E2]),E3]),
          exp(o(pd,Met2,Arg2),
              [E1,exp(o(pd,Met1,Arg1),[E2,E3])])).
regla2_1(exp(o(un,Met1,u([],Remp2)),
              [exp(o(un,Met2,u(Remp3,Remp4)),[E1,E2]),E3]),
          exp(o(un,Met2,u(Remp3,[])),
              [E1,exp(o(un,Met1,u(Remp4,Remp2)),[E2,E3])])).

/* distributividad de operadores binarios */
regla3(exp(o(jn,Met1,s(Arg)),
            [E1,exp(o(un,Met2,u(Remp1,Remp2)),[E2,E3])]),
        exp(o(un,Met2,u(Remp1,Remp2)),
            [exp(o(jn,Met1,s(Arg1)),[E1,E2]),
             exp(o(jn,Met1,s(Arg2)),[E1,E3])])):-
       invierte_remplazos(Remp1,R1),
       remplaza_condiciones(R1,Arg,Arg1),
       invierte_remplazos(Remp2,R2),
       remplaza_condiciones(R2,Arg,Arg2).
regla3(exp(o(pd,Met1,Arg1),
            [E1,exp(o(un,Met2,Arg2),[E2,E3])]),
        exp(o(un,Met2,Arg2),
            [exp(o(pd,Met1,Arg1),[E1,E2]),
             exp(o(pd,Met1,Arg1),[E1,E3])])).
regla3(exp(o(jn,Met1,s(Arg)),
            [exp(o(un,Met2,u(Remp1,Remp2)),[E2,E3]),E1]),
        exp(o(un,Met2,u(Remp1,Remp2)),
            [exp(o(jn,Met1,s(Arg1)),[E2,E1]),
             exp(o(jn,Met1,s(Arg2)),[E3,E1])])):-
       invierte_remplazos(Remp1,R1),
       remplaza_condiciones(R1,Arg,Arg1),
       invierte_remplazos(Remp2,R2),
       remplaza_condiciones(R2,Arg,Arg2).
regla3(exp(o(pd,Met1,Arg1),
            [exp(o(un,Met2,Arg2),[E2,E3]),E1]),
        exp(o(un,Met2,Arg2),
            [exp(o(pd,Met1,Arg1),[E2,E1]),
             exp(o(pd,Met1,Arg1),[E3,E1])])).

/* factorizacion de operadores binarios */
regla4(exp(o(un,Met2,u(Remp1,Remp2)),
            [exp(o(jn,Met1,s(Arg1)),[E1,E2]),
             exp(o(jn,Met1,s(Arg2)),[E1,E3])]),
        exp(o(jn,Met1,s(Arg)),
            [E1,exp(o(un,Met2,u(Remp1,Remp2)),[E2,E3])])):-
       remplaza_condiciones(Remp1,Arg1,Arg),
       remplaza_condiciones(Remp2,Arg2,A), Arg=A.
regla4(exp(o(un,Met2,Arg2),
            [exp(o(pd,Met1,Arg1),[E1,E2]),
             exp(o(pd,Met1,Arg1),[E1,E3])]),
        exp(o(pd,Met1,Arg1),
            [E1,exp(o(un,Met2,Arg2),[E2,E3])])).
regla4(exp(o(un,Met2,u(Remp1,Remp2)),
            [exp(o(jn,Met1,s(Arg1)),[E2,E1]),
             exp(o(jn,Met1,s(Arg2)),[E3,E1])]).
```

```
          exp(o(jn,Met1,s(Arg)),
              [exp(o(un,Met2,u(Remp1,Remp2)),[E2,E3]),E1])):-
          remplaza_condiciones(Remp1,Arg1,Arg),
          remplaza_condiciones(Remp2,Arg2,A), Arg=A.
  regla4(exp(o(un,Met2,Arg2),
              [exp(o(pd,Met1,Arg1),[E2,E1]),
               exp(o(pd,Met1,Arg1),[E3,E1])]),
          exp(o(pd,Met1,Arg1),
              [exp(o(un,Met2,Arg2),[E2,E3]),E1])).

  /* introduccion o eliminacion de una semijunta */
  regla5(exp(o(jn,Met1,s(Arg0)),[E1,E2]),
          exp(o(jn,Met1,s(Arg0)),
              [exp(o(sj,Met2,j(Arg1)),
                   [E1,exp(o(pj,Met3,p(Arg2)),[E3])]),E2])):-
          select(Arg1,Arg0),
          condicion_campos(Arg1,Campos),
          regla5_1(Campos,E2,E3),
          eval(base2,ev2,E1,A1),
A1=a2(Es1,_,r(Cr1,Er1),N1,_,S1,_,_,_,_),
          eval(base2,ev2,E3,A3),
A3=a2(Es3,_,r(Cr3,Er3),N3,_,S3,_,_,_,_),
          not(S1=S3),
          une_reduc(r([Arg1|Cr1],[a5(Es1,N1),a5(Es3,N3)|Er1]),
                    r(Cr3,Er3),W), W= r(Cr4,Er4),
          not(conjuntos_iguales(Cr1,Cr4)),
          es_aciclica(r(Cr4,Er4)),
          esquema_campos(Es3,Cs3),
          inter(Cs3,Campos,Arg2),
          un_metodo(sj,Met2),
          un_metodo(pj,Met3).
  regla5(exp(o(jn,Met1,s(Arg0)),[E1,E2]),
          exp(o(jn,Met1,s(Arg0)),
              [E1,exp(o(sj,Met2,j(Arg1)),
                      [E2,exp(o(pj,Met3,p(Arg2)),[E3])])])):-
          select(Arg1,Arg0),
          condicion_campos(Arg1,Campos),
          regla5_1(Campos,E1,E3),
          eval(base2,ev2,E2,A2),
A2=a2(Es2,_,r(Cr2,Er2),N2,_,S2,_,_,_,_),
          eval(base2,ev2,E3,A3),
A3=a2(Es3,_,r(Cr3,Er3),N3,_,S3,_,_,_,_),
          not(S2=S3),
          une_reduc(r([Arg1|Cr2],[a5(Es2,N2),a5(Es3,N3)|Er2]),
                    r(Cr3,Er3),W), W= r(Cr4,Er4),
          not(conjuntos_iguales(Cr2,Cr4)),
          es_aciclica(r(Cr4,Er4)),
          esquema_campos(Es3,Cs3),
          inter(Cs3,Campos,Arg2),
          un_metodo(sj,Met2),
          un_metodo(pj,Met3).

  regla5_1(_,E1,E1).
  regla5_1(Cs,exp(o(jn,_,_),[E1,E2]),E4):-
          eval(base0,ev0,E1,Atr1), Atr1=a0(Esq1,_),
```

```
          esquema_campos(Esq1,Cam1), inter(Cs,Cam1,[]),
regla5_1(Cs,E2,E4);
          eval(base0,ev0,E2,Atr2), Atr2=a0(Esq2,_),
          esquema_campos(Esq2,Cam2), inter(Cs,Cam2,[]),
regla5_1(Cs,E1,E4).

  /* Distribucion de operadores unarios respecto de binarios */
  regla6(exp(o(jn,Met1,s(Arg1)),[E1,E2]),
          exp(o(jn,Met1,s(Arg2)),
              [exp(o(sl,Met2,s(Arg3)),[E1]),E2])):-
         un_metodo(sl,Met2),
         eval(base0,ev0,E1,Atr1), Atr1= a0(Es1,_),
         esquema_campos(Es1,Camp1),
         divseleccion(Arg1,Camp1,Arg3,Arg2),
         not(Arg1=Arg2).
  regla6(exp(o(jn,Met1,s(Arg1)),[E1,E2]),
          exp(o(jn,Met1,s(Arg2)),
              [E1,exp(o(sl,Met2,s(Arg3)),[E2])])):-
         un_metodo(sl,Met2),
         eval(base0,ev0,E2,Atr2), Atr2= a0(Es2,_),
         esquema_campos(Es2,Camp2),
         divseleccion(Arg1,Camp2,Arg3,Arg2),
         not(Arg1=Arg2).

  regla6(exp(o(pj,Met1,p(Arg1)),
              [exp(o(jn,Met2,s(Arg2)),[E1,E2])]),
         exp(o(pj,Met1,p(Arg1)),
              [exp(o(jn,Met2,s(Arg2)),
                   [exp(o(pj,Met1,p(Arg3)),[E1]),E2])])):-
        eval(base0,ev0,E1,Atr1), Atr1= a0(Es1,_),
        condiciones_campos(Arg2,Campos),
        union(Arg1,Campos,Totales),
        esquema_campos(Es1,C1),
        inter(Totales,C1,Arg3).
  regla6(exp(o(pj,Met1,p(Arg1)),
              [exp(o(jn,Met2,s(Arg2)),[E1,E2])]),
         exp(o(pj,Met1,p(Arg1)),
              [exp(o(jn,Met2,s(Arg2)),
                   [E1,exp(o(pj,Met1,p(Arg3)),[E2])])])):-
        eval(base0,ev0,E2,Atr2), Atr2= a0(Es2,_),
        condiciones_campos(Arg2,Campos),
        union(Arg1,Campos,Totales),
        esquema_campos(Es2,C2),
        inter(Totales,C2,Arg3).
  regla6(exp(o(pj,Met1,p(Arg1)),[exp(o(pd,Met2,Arg2),[E1,E2])]),
         exp(o(pj,Met1,p(Arg1)),
              [exp(o(pd,Met2,Arg2),
                   [exp(o(pj,Met1,p(Arg3)),[E1]),E2])])):-
        eval(base0,ev0,E1,Atr1), Atr1= a0(Es1,_),
        esquema_campos(Es1,C1),
        inter(Arg1,C1,Arg3).
  regla6(exp(o(pj,Met1,p(Arg1)),[exp(o(pd,Met2,Arg2),[E1,E2])]),
         exp(o(pj,Met1,p(Arg1)),
              [exp(o(pd,Met2,Arg2),
                   [E1,exp(o(pj,Met1,p(Arg3)),[E2])])])):-
```

```
        eval(base0,ev0,E2,Atr2), Atr2= a0(Es2,_),
        esquema_campos(Es2,C2).
        inter(Arg1,C2,Arg3).

clauses
  aplica("?",Regla,E1,E2):- call(Regla,E1,E2), !.
  aplica("?",_,E1,E1).
  aplica("+",Regla,E1,E2):-
        call(Regla,E1,E3),
        aplica("*",Regla,E3,E2).
  aplica("*",Regla,E1,E2):-
        call(Regla,E1,E3), !,
        aplica("*",Regla,E3,E2).
  aplica("*",_,E1,E1).

  call(mejora1,E1,E2):- mejora1(E1,E2).
  call(mejora2,E1,E2):- mejora2(E1,E2).
  call(mejora3,E1,E2):- mejora3(E1,E2).
  call(mejora4,E1,E2):- mejora4(E1,E2).
  call(mejora5,E1,E2):- mejora5(E1,E2).
  call(mejora6,E1,E2):- mejora6(E1,E2).
  call(mejora7,E1,E2):- mejora7(E1,E2).
  call(regla1,E1,E2):- regla1(E1,E2).
  call(regla2,E1,E2):- regla2(E1,E2).
  call(regla3,E1,E2):- regla3(E1,E2).

predicates
  imple1(expresion,expresion,listainteger).
  imple2(expresion,expresion,listainteger).
  imple3(expresion,expresion,listainteger).

clauses
  /* cambio de metodo para la realizacion de una operacion */
  imple1(exp(o(Op,m(Met1,_),Args),Exps),
         exp(o(Op,Met2,Args),Exps),_):-
        metodo(Op,Met2), Met2=m(M2,_), not(Met1=M2).

  /* cambio de estrategia para transmision de relaciones */
  imple2(exp(o(pd,m(Met1,N1),Args),[E1,E2]),
         exp(o(pd,m(Met1,N2),Args),[E1,E2]),S):-
        select(N2,S), not(N1=N2).
  imple2(exp(o(jn,m(Met1,N1),Args),[E1,E2]),
         exp(o(jn,m(Met1,N2),Args),[E1,E2]),S):-
        select(N2,S), not(N1=N2).
  imple2(exp(o(un,m(Met1,N1),Args),[E1,E2]),
         exp(o(un,m(Met1,N2),Args),[E1,E2]),S):-
        select(N2,S), not(N1=N2).

  /* seleccion de una materializacion */
  imple3(rel(p(X1,X2,X3,X4,X5,[S|Ss],X7,X8)),
         rel(p(X1,X2,X3,X4,X5,[T,S|Tt],X7,X8)),_):-
        select2(Ss,T,Tt).

/* ------------------------------------------------- */
/* Elige una implementacion greedy                   */
```

```
domains
  dom_meta2 = dm2(operador,atrib).
  dom_meta2s= dom_meta2*.

predicates
  meta1(expresion,expresion).
  meta2(expresion,expresion).
  met2(expresion,expresion,atrib).
  elige_imp(operador,atribs,dom_meta2).
  minimum(dom_meta2s,operador,atrib).

clauses
  meta1(E1,E2):- mejora(E1,E3), !,
                 meta1(E3,E2).
  meta1(E1,E1).

  meta2(E1,E2):- met2(E1,E2,_).
  met2(exp(o(Op1,Met1,Arg1),[E1]),exp(Opr2,[E2]),Atr2):-
      met2(E1,E2,Atr1),
      findall(X,elige_imp(o(Op1,Met1,Arg1),[Atr1],X),L1),
      minimum(L1,Opr2,Atr2).
  met2(exp(o(Op1,Met1,Arg1),[E1,E2]),exp(Opr2,[E3,E4]),Atr3):-
      met2(E1,E3,Atr1),
      met2(E2,E4,Atr2),
      findall(X,elige_imp(o(Op1,Met1,Arg1),[Atr1,Atr2],X),L1),
      minimum(L1,Opr2,Atr3).
  met2(rel(R),rel(R),Atr):- base2(rel(R),Atr).

  elige_imp(o(Op,m(_,N),Arg),Atrs,
            dm2(o(Op,m(Met2,N),Arg),Atr2)):-
           metodo(Op,Met), Met=m(Met2,_),
           ev2(o(Op,m(Met2,N),Arg),Atrs,Atr2).

  minimum([dm2(Opr,Atr)],Opr,Atr):- !.
  minimum([dm2(_,a2(_,_,_,_,_,_,_,_,_,_,Costo))|Opers],
          Opr1,a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Co1)):-
         minimum(Opers,Opr1,
                 a2(E1,C1,R1,N1,S1,P1,O1,I1,L1,Co1)),
         Co1<=Costo, !.
  minimum([dm2(Opr,Atr)|_],Opr,Atr).

/* ------------------------------------------------ */
/* Reglas de generacion de expresiones              */
predicates
  genera2(integer,listainteger,expresion,expresion).

clauses
  proc_inicial(1,E1,E2):- meta1(E1,E2).
  proc_inicial(2,E1,E1).
  proc_inicial(3,E1,E2):- meta2(E1,E2).

  genera(_,E,E).
  genera(N,E1,E2):- sitio_usuario(S), genera2(N,[S],E1,E3),
meta1(E3,E2).
```

```
  genera2(0,S,E1,E2):- regla1(E1,E2); regla2(E1,E2);
regla3(E1,E2);
                       regla4(E1,E2); regla5(E1,E2);
imple1(E1,E2,S);
                       imple2(E1,E2,S); imple3(E1,E2,S).
  genera2(1,S,E1,E2):- regla3(E1,E2); regla4(E1,E2);
imple3(E1,E2,S).
  genera2(2,S,E1,E2):- regla1(E1,E2); regla2(E1,E2);
regla5(E1,E2);
                       imple2(E1,E2,S).
  genera2(3,S,E1,E2):- regla1(E1,E2); regla2(E1,E2);
regla6(E1,E2);
                       imple1(E1,E2,S).
  genera2(N,S,exp(Op,[E1]),exp(Op,[E2])):- genera2(N,S,E1,E2).
  genera2(N,S,exp(Op,[E1,E2]),exp(Op,[E3,E2])):-
         eval(base2,ev2,E2,A), A=a2(_,_,_,_,_,P,_,_,_,_),
         incl(P,S,Ss),
         genera2(N,Ss,E1,E3).
  genera2(N,S,exp(Op,[E1,E2]),exp(Op,[E1,E3])):-
         eval(base2,ev2,E1,A), A=a2(_,_,_,_,_,P,_,_,_,_),
         incl(P,S,Ss),
         genera2(N,Ss,E2,E3).
```

```
/* maquina.pro */

project "tesis2"
include "dominios.pro"


/* ============================================================
*/
/* Maquina que realiza una busqueda en el espacio de expresiones
*/
/*   usando una politica voraz
*/

predicates
  search1(listainteger,expresion,expresion,
          expresion,integer,integer,real).
  search2(integer).
  expande(integer,expresion).
  procesa(nodo).
  terminacion.
  tiempo_ejecucion(real,real).
  tiempo_optimizacion(real).

clauses
  search(Z,E1,E2):- search1(Z,E1,E2,E1,0,0,0).

  search1([A|Z],E1,E2,E0,M,N,T):-
         proc_inicial(A,E1,E3),
         exp_costo(E3,C3),
         asserta(abierto([n(C3,E3)])),
         asserta(mejor(n(C3,E3))),
         time(0,0,0,0),
         gensym(n,_),
         gensym(m,_),
         search2(A), nl,
         contador(n,N1),
         contador(m,M1),
         tiempo_optimizacion(T1),
         retract(abierto(_)),
         reset_gensym(n),
         reset_gensym(m),
         retract(mejor(n(_,E4))), !,
         N2= N+N1, M2= M+M1, T2= T+T1,
         search1(Z,E4,E2,E0,M2,N2,T2).

  search1([],E1,E1,E0,M,N,T):-
         exp_costo(E0,C0),
         exp_costo(E1,C1),
         tiempo_ejecucion(C0,T0),
         tiempo_ejecucion(C1,T1),
         X= T0/T1,
         writedevice(WD),
         abreasch(out1,"salida.dat"),
         nl,
         print(E0),
```

```
        nl,
        write("metodo VORAZ\n"),
        writef("% expresiones generadas.\n",M),
        writef("% expresiones expandidas.\n",N),
        writef("%1.2 segundos de ejecucion exp inicial.\n",T0),
        writef("%1.2 segundos de ejecucion de solucion.\n",T1),
        writef("reduccion de %1.2 : 1.\n",X),
        writef("% segundos de optimacion.\n",T),
        nl,
        print(E1), nl,
        empaca(E1,I1),
        programa_ejecucion(I1),
        nl,
        destruye_mesh,
        write("-----------------------------\n\n"),
        closefile(out1),
        writedevice(WD),
        nl,
        write("metodo VORAZ\n"),
        writef("reduccion de %1.2 : 1.\n",X),
        writef("% segundos de optimacion.\n",T),
        nl.

search2(Z):- retract(abierto([n(_,E)|])),
             asserta(abierto([])),
             write('*'),
             expande(Z,E), terminacion, !.
search2(_).

expande(Z,Nodo):- gensym(n,_),
                  genera(Z,Nodo,E2),
                  gensym(m,_),
                  exp_costo(E2,C2),
                  procesa(n(C2,E2)),
                  terminacion.
expande(_,_).

procesa(n(C1,E1)):- mejor(n(C,_)), C1<C, write('+'),
                    retract(mejor(_)),
asserta(mejor(n(C1,E1))),
                    retract(abierto(_)),
                    asserta(abierto([n(C1,E1)|])), !.

terminacion:- storage(_,S,_), S<150000, !.
terminacion:- tiempo_optimizacion(T),
              limite(L),
              L<=T.

tiempo_ejecucion(Accesos,Tiempo):-
                 tiempo_acceso(TA),
                 Tiempo= Accesos*TA.
tiempo_optimizacion(Tiempo):-
                    time(Horas,Minutos,Segundos,Centesimas),
                    Tiempo=
3600*Horas+60*Minutos+Segundos+Centesimas/100.
```

```
/* maquina2.pro */

project "tesis2"
include "dominios.pro"


/* ================================================================
*/
/* Maquina que realiza una busqueda en el espacio de expresiones
*/
/*   usando una politica de ascenso de la colina
*/

predicates
  search1(listainteger,expresion,expresion,
          expresion,integer,integer,real).
  search2(integer).
  expande(integer,expresion).
  procesa(nodo).
  agrega(nodo,nodos,nodos).
  terminacion.
  limpia_cerrado.
  tiempo_ejecucion(real,real).
  tiempo_optimizacion(real).

clauses
  search(Z,E1,E2):- search1(Z,E1,E2,E1,0,0,0).

  search1([A|Z],E1,E2,E0,M,N,T):-
        proc_inicial(A,E1,E3),
        exp_costo(E3,C3),
        tiempo_ejecucion(C3,T3),
        inversion_opt(FI),
        L3= FI*T3,
        asserta(limite(L3)),
        asserta(abierto([n(C3,E3)])),
        asserta(mejor(n(C3,E3))),
        time(0,0,0,0),
        gensym(n,_),
        gensym(m,_),
        search2(A), nl,
        contador(n,N1),
        contador(m,M1),
        tiempo_optimizacion(T1),
        retract(abierto(_)),
        retract(limite(_)),
        reset_gensym(n),
        reset_gensym(m),
        limpia_cerrado,
        retract(mejor(n(_,E4))), !,
        N2= N+N1, M2= M+M1, T2= T+T1,
        search1(Z,E4,E2,E0,M2,N2,T2).

  search1([],E1,E1,E0,M,N,T):-
        exp_costo(E0,C0),
```

```
        exp_costo(E1,C1),
        tiempo_ejecucion(C0,T0),
        tiempo_ejecucion(C1,T1),
        X= T0/T1,
        hill_climbing(HC),
        writedevice(WD),
        abrearch(out1,"salida.dat"),
        nl,
        print(E0),
        nl,
        writef("metodo ASCENSO-DE-COLINA(%)\n",HC),
        writef("% expresiones generadas.\n",M),
        writef("% expresiones expandidas.\n",N),
        writef("%1.2 segundos de ejecucion exp inicial.\n",T0),
        writef("%1.2 segundos de ejecucion de solucion.\n",T1),
        writef("reduccion de %1.2 : 1.\n",X),
        writef("% segundos de optimacion.\n",T),
        nl,
        print(E1), nl,
        empaca(E1,I1),
        programa_ejecucion(I1),
        nl,
        destruye_mesh,
        write("-------------------------------\n\n"),
        closefile(out1),
        writedevice(WD),
        nl,
        writef("metodo ASCENSO-DE-COLINA(%)\n",HC),
        writef("reduccion de %1.2 : 1.\n",X),
        writef("% segundos de optimacion.\n",T),
        nl.

search2(Z):- retract(abierto([n(_,E)|Abierto])),
             asserta(abierto(Abierto)),
             asserta(cerrado(E)),
             write('*'),
             expande(Z,E), terminacion, !.
search2(_).

expande(Z,Nodo):- gensym(n,_),
                  genera(Z,Nodo,E2),
                  not(cerrado(E2)),
                  gensym(m,_),
                  exp_costo(E2,C2),
                  procesa(n(C2,E2)),
                  terminacion.
expande(_,_).

procesa(n(C1,E1)):- mejor(n(C,_)), C1<C, write('+'),
                    retract(mejor(_)),
asserta(mejor(n(C1,E1))),
                    retract(abierto(Abierto)),
                    asserta(abierto([n(C1,E1)|Abierto])),
                    retract(limite(L1)),
                    tiempo_ejecucion(C1,TE),
```

```
                        tiempo_optimizacion(T0),
                        inversion_opt(FI),
                        L2= T0+FI*TE,
                        min(L1,L2,L3),
                        asserta(limite(L3)), !.
procesa(n(C1,E1)):- mejor(n(C,_)), hill_climbing(F), C1<=F*C,
                        retract(abierto(Abierto)),
                        agrega(n(C1,E1),Abierto,Abierto2),
                        asserta(abierto(Abierto2)), !.

agrega(Nodo,[],[Nodo]):- write('.').
agrega(Nodo,[Nodo|Abierto],[Nodo|Abierto]):- !.
agrega(n(C1,I1),[n(C2,I2)|Abierto],
       [n(C1,I1),n(C2,I2)|Abierto]):-
      C1<C2, write('.'), !.
agrega(Nodo1,[Nodo2|Abierto],[Nodo2|Abierto2]):-
      agrega(Nodo1,Abierto,Abierto2).

terminacion:- storage(_,S,_), S<150000, !.
terminacion:- fail, tiempo_optimizacion(T),
              limite(L),
              L<=T.

limpia_cerrado:- retract(cerrado(_)), fail.
limpia_cerrado.

tiempo_ejecucion(Accesos,Tiempo):-
                  tiempo_acceso(TA),
                  Tiempo= Accesos*TA.
tiempo_optimizacion(Tiempo):-
                     time(Horas,Minutos,Segundos,Centesimas),
                     Tiempo=
3600*Horas+60*Minutos+Segundos+Centesimas/100.
```

```
/* dialogo.pro */

project "tesis2"
include "dominios.pro"


predicates
  interpreta(string).
  opcion(char,char,string).
  printcosto(expresion).
  go.

clauses
  interpreta("fin"):- !.
  interpreta(""):- !, fail.
  interpreta(Comando):- frontchar(Comando,Var,S1),
frontchar(S1,Op,Exp),
                        opcion(Op,Var,Exp), nl, !, fail.
  interpreta(_):- write("Error, comando invalido."), nl, fail.

  opcion('=',Var,SExp):- parse(SExp,Exp1), print(Exp1),
                         asigna(Var,Exp1).
  opcion('?',Var,_)   :- variable(Var,Exp1), print(Exp1),
printcosto(Exp1).
  opcion('!',Var,SExp):- parse(SExp,Exp1),
                         search([1,2,3],Exp1,Exp2),
                         print(Exp2), printcosto(Exp2),
                         asigna(Var,Exp2).
/*  opcion('1',Var,SExp):- parse(SExp,Exp1),
                         search([1],Exp1,Exp2),
                         print(Exp2), printcosto(Exp2),
                         asigna(Var,Exp2).
  opcion('2',Var,SExp):- parse(SExp,Exp1),
                         search([2],Exp1,Exp2),
                         print(Exp2), printcosto(Exp2),
                         asigna(Var,Exp2).
  opcion('3',Var,SExp):- parse(SExp,Exp1),
                         search([3],Exp1,Exp2),
                         print(Exp2), printcosto(Exp2),
                         asigna(Var,Exp2).*/
  opcion('>',Var,_)   :- variable(Var,Exp1),
                         writedevice(WD),
                         abrearch(out1,"salida.dat"),
                         nl,
                         print(Exp1),
                         nl,
                         empaca(Exp1,I1),
                         programa_ejecucion(I1),
                         nl,
                         destruye_mesh,
                         write("-----------------------------
\n\n"),
                         closefile(out1),
                         writedevice(WD).
```

```
  printcosto(Exp):- exp_costo(Exp,C), ceil(C,C1), write("Costo:
",C1), nl.

  go:- repeat, storage(_,B,_), write(B),
       write('>'), readln(Comando), interpreta(Comando), !.

goal
  inicializa, clearwindow, nl,
  write("***** Optimacion de consultas basada en reglas V2
(Agosto 1987)."),
  nl, nl, go, clearwindow.
```

Bibliografía

[1] ACM SIGMOD
Proceedings of the 1987 Annual Conference
San Francisco, Mayo 27-29, 1987.

[2] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman.
Data Structures and Algorithms
Addison-Wesley, Reading Mass., 1983.

[3] Alfred V. Aho, Jeffrey D. Ullman.
Principles of Compiler Design
Addison-Wesley, Reading Mass., 1977.

[4] P. M. G. Apers, A. L. Hevner, S. B. Yao.
"Optimization algorithms for distributed queries"
IEEE Transactions on Software Engineering
Vol SE-9, No 1, Enero 1983, pp. 57-68.

[5] M. M. Astrahan, D. D. Chamberlin.
"Implementation of a structured english query language"
Communications of the ACM
Vol 18, No 10, Octubre 1975, pp. 580-588.

[6] Renato Barrera, Aniceto Saboya.
"Data structure support for a geographic database management system"
IEEE CAPAIDM 1985, pp. 359-366.

[7] Catriel Beeri, et al.
"On the desirability of acyclic database schemes"
Journal of the ACM
Vol 30, No 3, Julio 1983, pp. 479-513.

[8] Philip. A. Bernstein, D. M. Chiu.
"Using semi-joins to solve relational queries"
Journal of the ACM
Vol 28, No 1, Enero 1981, pp. 25-40.

[9] Philip A. Bernstein, et al.
"Query processing in a system in a system for distributed databases (SDD-1)"
ACM Transactions on Database Systems
Vol 6, No 4, Diciembre 1981, pp. 602-625.

[10] Borland International, Inc.
Turbo-Prolog Owner's Handbook, 1986.

[11] Michael J. Carey, et al.
"The architecture of the EXODUS extensible DBMS"
Proceedings of the International Workshop on Object-Oriented Database Systems, 1986, pp. 18-25.

[12] Stefano Ceri, Giuseppe Pelagatti.
"Correctness of query execution strategies in
distributed databases"
ACM Transactions on Database Systems
Vol 8, No 4, 1983.

[13] Stefano Ceri; Giuseppe Pelagatti.
Distributed Databses: Principles and Systems
McGraw-Hill Book Company, New York, 1985.

[14] W. Clocksin, C. Mellish.
Programming in Prolog
Springer-Verlag, New York, 1981.

[15] Edgar F. Codd.
"A relational model of data for large shared data banks"
Communications of then ACM
Vol 13, No 6, Junio 1970, pp. 377-387.

[16] Edgar F. Codd.
"Relational completeness of data base sublanguages"
en R. Rustin Database Systems, pp 65-98.
Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[17] Edgar F. Codd.
"Extending the database relational model
to capture more meaning"
ACM Tansactions on Database Systems
Vol 4, No 4, Diciembre 1979, pp. 397-434.

[18] J. A. Darlington, R. M. Burstall.
"A system which automatically improves programs"
Acta Informatica
No 6, 1976, pp. 41-60.

[19] Umeshwar Dayal, et al.
"PROBE: A research project in knowledge-oriented
database systems: Preliminary analysis"
Technical Report CCA-85-03
Computer Coprporation of America, Cambridge MA., 1985.

[20] Edsger W. Dijkstra.
A Discipline of Programming
Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[21] Johan C. Freytag.
"A rule-based view of query optimization"
ACM-SIGMOD, 1987.

[22] Johan C. Freytag, Nathan Goodman.
"Translating aggregate queries into iterative programs"
VLDB, 1986.

[23] Goetz Graefe; David J. DeWitt.
"The EXODUS optimizer generator"
Proceedings of the ACM-SIGMOD, 1987.

[24] A. L. Hevner, S. B. Yao.
"Query processing in distributed database systems"
IEEE Transactions on Software Engineering
Vol SE-5, No 3, Mayo 1979, pp. 177 187.

[25] Yannis E. Ioannidis; Eugene Wong.
"Query optimization by simulated annealing"
Proceedings of the ACM-SIGMOD, 1987.

[26] Matthias Jarke; Jurgen Koch.
"Query optimization in database systems"
ACM Computer Surveys
Vol 16, No 2, Junio 1984, pp. 111-152.

[27] Yahiko Kambayashi
"Processing of ciclic queries"
en Kim et al. Query Processing
Springer-Verlag, New York, 1986, pp. 62-78.

[28] Jonathan J. King.
"QUIST: A system for semantic query optimization"
en VLDB, 1982.

[29] Frank Manola, Umeshwar Dayal.
"PDM-An object-oriented data model"
Proceedings of the International Workshop on
Object-Oriented Database Systems, 1986, pp. 18-25.

[30] David Reiner, Arnon Rosenthal.
"Strategy spaces and abstract target machines
for query optimization"
en Kim et al. Database Engineering Vol 1, pp. 228-223.
IEEE Computer Science Press, Silver Spring, MD, 1983.

[31] E. Rich.
Artificial Intelligence
McGraw-Hill, New York, N. Y., 1983.

[32] David Shipman.
"The functional model and the data language DAPLEX"
ACM Transactions on Database Systems
Vol 6, No 1, Marzo 1981, pp. 140-173.

[33] John M. Smith; Philip Y. T. Chang.
"Optimizing the performance of a relational algebra
database interface"
Communications of the ACM
Vol 18, No 10, Octubre 1975, pp. 568-579.

[34] Jeffrey D. Ullman.
Principles of Database Systems
Segunda edicion
Computer Science Press, Rockville Md., 1982.

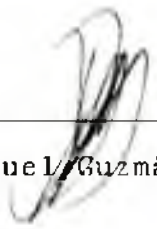
[35] Patrick H. Winston.
Artificial Intelligence
Segunda edicion
Addison-Wesley, Reading Mass., 1984.

[36] S. Bing Yao
"Approximating block accesses in database organizations"
Communications of the ACM
Vol 20, No 4, Abril 1977, pp. 260-261.

[37] C. T. Yu; C. C. Chang.
"Distributed query processing"
ACM Computing Surveys
Vol 16, No 4, Diciembre 1984.

El jurado designado por la Sección de Computación del Departamento de Ingeniería Eléctrica del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó esta tesis el 25 de Septiembre de 1987.

Dr. Renato Barrera Rivera



Dr. Manuel Guzmán Rentería

Dr. Zdenek Zdrahal