BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

"IMPLANTACIÓN DEL MÉTODO DE FILTRADO ESTÁTICO EN UNA BASE DE DATOS DEDUCTIVA"

TESIS PROFESIONAL

Que para obtener el grado de Licenciado

en Ciencias de la Computación

Presenta

Blanca del Carmen Naredo Lara

ASESORADA POR :

Dra. María Josefa Somodevilla García

Junio 2009

Contenido

Introducción **I**

1 Modelo Relaciona **1**

1.1 La Estructura de Datos Relaciónales 2

1.2 Las Reglas de Integridad Rélacional 5

1.3 La Regla de Integridad de las Entidades 5

1.4 Parte Manipulativa 7

1.4.1 Operaciones Tradicionales 7

1.4.2 Operaciones Adicionales Especiales 9

1.5 Lenguaje de Consulta (SQL) 11

1.5.1 Estructuras Básicas 12

1.5.2 La Potencialidad de *SQL* 13

1.6 Limitaciones del Modelo Rélacional 13

2 Técnicas de Consulta para Bases de Datos Deductivas **15**

2.1 Sintaxis de las Bases de Datos Deductivas 16

2.1.1 Prolog y Bases de Datos 17

2.2 Técnicas de Consulta 18

2.3 Métodos de Evaluación 19

2.4 Métodos de Reescritura 21

2.4.1 Métodos 21

3 Implantación del Método de Filtrado Estático **31**

3.1 Descripción del Método 31

3.2 Descripción de la Base de Datos 32

3.3 Pseudocódigo del Método de Filtrado Estático..... 34

CONCLUSIONES **35**

A SQL **36**

B Programa **46**

C Ejemplos **50**

Introducción

Las **Bases de Datos Relaciónales** fueron diseñadas principalmente para aplicaciones que procesan grandes volúmenes de datos, de formato fijo y relativamente simples.

Debido al fundamento matemático del Modelo de Datos Relacional y a su simplicidad, fue predominante durante los años de la década de los ochentas y continúa en la década de los noventas.

Los Sistemas de Bases de Datos tradicionales presentan problemas en la optimización del código y en la habilidad del sistema para ejecutar la consulta u otras operaciones sobre los datos.

Debido a los problemas presentados surgen las Bases de Datos Deductivas y los Sistemas Manejadores de Bases de Datos Orientados a Objetos.

Los **Sistemas de Bases de Datos Deductivas** son sistemas manejadores de bases de datos, formados por un lenguaje de consulta y una estructura de almacenamiento basada en un modelo lógico de datos, el cuál consiste de una notación matemática para describir los datos y las relaciones y una técnica que facilita las Consultas y verifica el cumplimiento de las Restricciones de Integridad.

Una Base de Datos Deductiva es un conjunto de reglas no ordenadas, la cual está particionada en un conjunto de hechos y en un conjunto de todas las reglas.

- **Objetivo General.** Explorar el comportamiento de los Métodos de Reescri tura (Conjuntos Mágicos, Conteo, Filtrado Estático) de las Bases de Datos Deductivas.
- **Objetivo Particular.** Implantación y Análisis del funcionamiento del Método de Filtrado Estático, mostrando sus características y ventajas principales.
- Para dar cumplimiento a los objetivos el trabajo se estructura en:

  - **Capítulo 1.** Describe el desarrollo histórico, la estructura y las limitaciones del Modelo Relacional, se realiza una presentación de las principales características y ejemplos del Algebra Relacional y por último se menciona al Lenguaje de Consulta SQL.
  - **Capítulo 2.** Presenta una introducción de las Bases de Datos Deductivas y de los Métodos de Evaluación (Ingenua, SemiIngenua y Consulta_Subconsulta).

    Se describen los Métodos de Reescritura (Conjuntos Mágicos, Conteo y Filtrado Estático), se muestran los algoritmos y el desarrollo de ejemplos de cada uno de ellos.
  - **Capítulo 3.** Se implantó el Método de Filtrado Estático previo a una breve descripción del Método y de la Base de Datos.

Para finalizar se resume el trabajo presentado en las Conclusiones del mismo.

Capítulo 1

Modelo Relacional

Los primeros Sistemas de Bases de Datos estaban basados en el Modelo de Red o en el Modelo de Datos Jerárquico, ambos están más ligados a la implementación física de la Base de Datos que el Modelo Relacional. El Modelo de Datos Relacional es relativamente nuevo, se fundamenta en el Algebra Relacional, el cual ayuda al diseño de las Bases de Datos Relaciónales y al procesamiento eficiente de solicitudes de información por parte de los usuarios; además cuenta con bases sólidas en aspectos de la matemática. El modelo relacional se presenta como el principal modelo de datos para aplicaciones comerciales de procesamiento de datos. Este representa la tendencia dominante en el mercado actual a pesar de que nuevos modelos de datos como el Orientado a Objetos y el Lógico están ganando terreno.

En 1970, Codd [Codd 1970], propone el uso del modelo relacional para representar a los Sistemas de Bases de Datos. Este modelo se basa en relaciones n-arias y en la normalización de dichas relaciones.

El Modelo Relacional se ocupa de tres aspectos de los datos:

- Estructura
- Integridad
- Manipulación

1.1 La Estructura de Datos Relaciónales

Una Base de Datos Relacional (BDR) consiste de una colección de tablas, a cada una se le asigna un nombre único, donde los operadores generan tablas nuevas a partir de las ya existentes.

A continuación se definen los conceptos relacionados con la estructura de las BDR:

Conceptos relacionados con la Estructura de las BDR.

- **Relación**

  Una relación corresponde a lo que se llama comúnmente una tabla. Codd introduce el término de relación adoptando el concepto matemático como un subconjunto del producto cartesiano de una lista de dominios, es decir, dados los conjuntos D_1, D_2,...,D_n (no necesariamente distintos), R es una relación sobre esos n conjuntos si es un conjunto de n-tuplas donde cada una tiene como primer elemento un D_1, como segundo un D_2 y así sucesivamente.

  Una relación se compone de dos partes una cabecera y un cuerpo.

  - **Cabecera**

    Está formada por un conjunto fijo de atributos, es decir, de pares atributo-dominio, $\{(A_1 : D_1), (A_2 : D_2),..., (A_n : D_n)\}$ tales que cada atributo Aj corresponda a uno y sólo uno de los dominios subyacentes $Dj (j = 1, 2,.., n)$.

  - **Cuerpo**

    Está formado por un conjunto de tuplas, el cual varía con el tiempo. Cada tupla a su vez está formada por un conjunto de pares; atributo-valor $\{(A_1 : V_{i1}), (A_2 : V_{i2}), \ldots, (A_n : V_{in})\}$ $(i = 1, ...,m)$ donde m es el número de tuplas del conjunto. En cada una de estas tuplas hay uno de

estos pares atributo-valor $(A_j : V_{ij})$ para cada atributo A_j de la cabecera. Para cada par de atributo-valor $(A_j : V_{ij})$ donde V_{ij} es un valor del dominio único D_j asociado al atributo A_j.

Término Formal	Término Informal
Relación	Tabla
Tupla	Fila o Registro
Atributo	Columna o campo
Cardinalidad	Número de filas
Grado	Número de columnas

Tabla 1. Terminología de la Estructura de Datos Relacional.

Las Relaciones poseen cuatro propiedades importantes:

- Las tuplas no están ordenadas
- No existen tuplas duplicadas
- Los atributos no están ordenados
- Todos los valores de los atributos simples son atómicos

Una relación esta normalizada si en cada posición de fila y columna dentro de una tabla, existe siempre un solo valor nunca una lista de valores.

Dominio

Se define como un conjunto de valores escalares todos del mismo tipo, o bien, como una colección de valores de los cuales uno o más atributos (columnas) obtienen sus valores reales.

Los valores escalares representan **la menor unidad semántica de información** en el sentido de que son atómicos, es decir, no poseen estructura interna. Un dominio es atómico si sus elementos se consideran unidades indivisibles.

- **Dominio Simple**
  Se consideran aquellos que contienen valores atómicos, por ejemplo "integer", "real", etc.

- **Dominio Compuesto**
  Se consideran aquellos que son combinación de dominios simples, se define sobre el producto cartesiano de los dominios, por ejemplo, considere un tipo llamado "fecha": mes/día/año. La importancia fundamental de los dominios es que **restringen las comparaciones.**

A continuación se mencionan algunos tipos de relaciones que existen en un Sistema Relacional.

- **Relación Base (Relaciones reales)**
  Es una relación autónoma con nombre. Es decir son aquellas cuya importancia es tal que se ha decidido darles un nombre y hacerlas parte directa de la BD.

- **Vistas (Relaciones virtuales)**
  Es una relación derivada con nombre, representada mediante su definición en términos de otras relaciones con nombre; no posee datos almacenados propios, separados y distinguibles (a diferencia de las relaciones base).

- **Instantáneas (Relaciones derivadas)**
  También es una relación derivada, con nombre como una vista, pero a diferencia de las vistas, las instantáneas son reales, no virtuales, es decir, están representadas no sólo por su definición en términos de otras relaciones con nombre, sino también por sus propios datos almacenados.

Otros tipos de relaciones son resultados de consulta, resultados intermedios y las relaciones temporales.

1.2 Las Reglas de Integridad Relacional

El propósito de las reglas de integridad es informar al Sistema Manejador de Base de Datos (DBMS) de ciertas restricciones del mundo real. El modelo relacional incluye dos reglas generales, a las llaves primarias y a las llaves ajenas.

- **Llave Primaria**
  Se considera como un identificador único para una relación. La relación puede tener varios identificadores a los cuales se les llama **Llaves candidatas,** se debe escoger sólo una como llave primaria.

  Las llaves candidatas deben cumplir las siguientes propiedades:

  1. **Unicidad.-** En cualquier momento dado no existen dos tuplas en R con el mismo valor de K.

  2. **Minimalidad.-** Si K es compuesto, no será posible eliminar ningún componente de K sin destruir la propiedad de unicidad.

1.3 La Regla de Integridad de las Entidades

La Regla de Integridad de las Entidades se expresa:

"En una Base de Datos Relacional nunca se registra información acerca de algo que no podemos identificar".

"Ningún componente de la llave primaria de una relación base puede aceptar valores nulos".

- **Llave ajena**

  De manera informal una llave ajena es un atributo (quizá compuesto) de una relación cuyos valores deben concordar con los de la llave primaria de alguna relación. Representa una referencia a la tupla donde se encuentra el valor correspondiente de la llave primaria (la tupla referida o tupla objetivo).

- **Regla de Integridad Referencial**

  "La base de datos no debe contener valores de llave ajena sin concordancia".

  La regla de integridad referencial tan solo dice que si B hace referencia a A entonces A debe existir.

  La relación que contiene a la llave ajena se conoce como **relación referencial** y la relación que contiene a la llave primaria correspondiente se denomina **relación referida o relación objetivo.**

  La justificación de esta regla es la siguiente: Así como los valores de la llave primaria representan identificadores de entidades, así los valores de la llave ajena representan referencias a entidades (a menos de que sean nulos).

1.4 Parte Manipulativa

La parte manipulativa, se divide en dos partes:

1. Conjunto de operadores, como los de reunión forman un conjunto, los llamados del "Algebra Relacional".
2. Operaciones de Asignación, asigna el valor de alguna expresión arbitraria del algebra de una relación asombrada.

- **Algebra Relacional**
  El álgebra relacional es un lenguaje de consulta procedimental, consiste de un conjunto de operadores de alto nivel que operan sobre relaciones. Las operaciones tradicionales en el álgebra relacional son **selección, proyección, producto natural, unión y diferencia de conjuntos.** Además **intersección de conjuntos, producto cartesiano, división y asignación.**

1.4.1 Operaciones Tradicionales

- **Selección**
  Selecciona tuplas que satisfacen un predicado dado. Se utiliza la letra griega minúscula σ para indicar la selección. El predicado aparece como subíndice de σ.

  Ejemplo 1

  $$\sigma_{producto = "CompIBM"}(Proveedores)$$

  En el predicado de selección se permite el manejo de comparaciones usando $=, \neq, <, \leq, >, \geq$. Además, pueden combinarse varios predicados en un predicado más complejo usando los conectores *y*(Λ) y o(V).

El predicado de selección puede incluir comparaciones entre dos atributos.

Ejemplo 2

$$\sigma_{producto = \text{“}diskettes5\text{”} \Lambda\ precio > 30}\ (Proveedores)$$

- **Proyección**

  Nos permite evitar la redundancia, es una operación unitaria que devuelve su relación argumento con ciertas columnas omitidas. La proyección se indica por la letra griega Π, se listan los atributos que se desea que aparezcan en el resultado como subíndice de Π, la relación argumento se escribe a continuación entre paréntesis. Puesto que una relación es un conjunto, se eliminan todas las filas duplicadas.

Ejemplo 3

$$\Pi_{nombre.saldo}\ (Clientes)$$

- **Producto Natural**

  Es una operación binaria que nos permite combinar ciertas selecciones y un producto cartesiano en una operación. Esta operación forma un producto cartesiano de sus dos argumentos, realiza una selección forzando la igualdad en aquellos atributos que aparezcan en ambas planificaciones de relaciones y quita las columnas duplicadas. Se representa por el símbolo de producto | x |

Ejemplo 4

$$\Pi_{producto.precio.cantidad}\ (\ Proveedores\ \ |\ x\ |\ \ Envíos\ )$$

Unión

Construye una relación formada por todas las tuplas que aparecen en cualquiera de las dos relaciones especificadas. Esta operación está definida para relaciones con la misma aridad, se representa por U.

Ejemplo 5

$\Pi_{nombre}(\sigma_{producto = \text{“diskettes5”}}(Proveedores)) \cup \Pi_{nombre}(\sigma_{producto = \text{“diskettes5”}}(Pedidos))$

Diferencia de Conjuntos

Nos permite encontrar tuplas que estén en una relación pero no en otra. Se representa por - .

Ejemplo 6

$\Pi_{nombre}(\sigma_{producto = \text{“diskettes5”}}(Pedidos)) - \Pi_{nombre}(\sigma_{producto = \text{“diskettes5”}}(Proveedores))$

1.4.2 Operaciones Adicionales Especiales

Las operaciones tradicionales del álgebra relacional son suficientes para expresar cualquier consulta en álgebra relacional. Como algunas consultas comunes son largas de expresar, se definen operaciones adicionales que no añaden ninguna potencia al álgebra, pero que simplifican consultas comunes.

- **Intersección de Conjuntos**

  Se construye una relación formada por aquellas tuplas que pertenezcan a las dos relaciones especificadas. Esta operación se representa por el símbolo ∩.

Ejemplo 7

$$\Pi_{nombre}(\sigma_{producto = \text{“diskettes5”}}(Proveedores)) \cap \Pi_{nombre}(\sigma_{producto = \text{“diskettes5”}}(Pedidos))$$

- **Producto Cartesiano**

Esta operación permite combinar información de varias relaciones, es una operación binaria, la cual esta representada por una cruz *x* .

Ejemplo 8

$$\Pi_{nombre} = \text{"Teresa Contreras"}\ (Clientes \times Pedidos)$$

- **Operación División**

Se establece para aquellas consultas que incluyen la frase *para todos,* esta representada por el símbolo $\div$.

- **Operación Asignación**

Algunas veces es conveniente escribir una expresión por partes usando la asignación a una variable de relación temporal, funciona de forma parecida a la asignación en un lenguaje de programación convencional, se representa por ($\leftarrow$)

1.5 Lenguaje de Consulta (SQL)

SQL ("Structured Query Languaje", Lenguaje de Consulta Estructurado) es una combinación de constructores del álgebra relacional y del cálculo relacional. Incluye características para definir la estructura de datos, para modificar los datos de la BD y para especificar las restricciones de seguridad.

Es un lenguaje con origen en los Laboratorios de Investigación de IBM en San José, originalmente fue llamado Seguel. Este se implemento como parte del proyecto del Sistema R en los primeros años de la década de los setenta y en la actualidad numerosos productos soportan este lenguaje. En 1986, el Instituto Nacional Americano de Estándares (ANSI) publicó un SQL estándar y la Organización Internacional para Estandarización (ISO) lo reconoció como el lenguaje estándar oficial para interactuar con sistemas de Bases de Datos Relacionales.

SQL consta de las siguientes partes:

- **Lenguaje de Definición de Datos (DDL),** proporciona órdenes para definir esquemas de relación, eliminar relaciones, crear índices y modificar esquemas de relación.
- **Lenguaje de Manipulación de Datos Interactivo (DML)**, incluye un lenguaje de consulta basado en el álgebra relacional y el cálculo relacional de tuplas. Además comprende comandos para insertar, suprimir y modificar tuplas de la Base de Datos.
- **Lenguaje de Manipulación de Datos Inmerso (DML)**, está diseñado para usarse dentro de los lenguajes de programación corno PL/I, Cobol, Pascal, Fortran y C.
- **Definición de Vistas,** el SQL DDL incluye órdenes para definir vistas.
- **Automatización,** DDL incluye órdenes para especificar derechos de acceso a relaciones y vistas.

- **Integridad,** versiones recientes de SQL proporcionan únicamente una forma limitada de comprobación de integridad.
- **Control de Transacciones,** SQL incluye órdenes para especificar el comienzo y final de las transacciones.

1.5.1 Estructuras Básicas

- La estructura básica de una expresión en SQL consta de tres cláusulas **select, from y where**

  select corresponde a la operación de proyección del álgebra relacional, lista los atributos que se desean en el resultado de una consulta.

  from corresponde a la operación de producto cartesiano del álgebra relacional, lista las relaciones que se van a examinar en la evaluación de la expresión.

  where corresponde al predicado de selección del álgebra relacional, consta de un predicado que implica atributos de las relaciones que aparecen en la cláusula from.

  Una consulta típica en SQL tiene la forma:

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

  Cada A_i representa un atributo, cada r_i una relación y P es un predicado. Esta consulta es equivalente a la expresión del álgebra relacional.

  Si se omite la cláusula **where,** el predicado P es verdadero. La lista de atributos puede sustituirse por asterisco (*) el cual selecciona todos los atributos.

El resultado de una consulta en SQL es una relación.

En el apéndice A se presentan ejemplos que ilustran la manipulación de los datos utilizando *SQL*.

1.5.2 La Potencialidad de *SQL*

SQL es tan poderoso en expresividad como el álgebra relacional porque *SQL* incluye las operaciones fundamentales del álgebra relacional. La proyección se realiza en la cláusula **select,** el producto cartesiano se representa por medio de la cláusula **from** y los predicados de selección del álgebra se representan en la cláusula **where.** El álgebra relacional y *SQL* incluyen la unión y la diferencia.

SQL ofrece una rica colección de características, que incluyen funciones de agregación, ordenamiento de tuplas y otras capacidades.

Muchas implementaciones de SQL permiten hacer consultas en SQL desde un programa escrito en un lenguaje como Pascal, PL/I, Fortran, C o Cobol.

1.6 Limitaciones del Modelo Relacional

La simplicidad y el fundamento matemático hicieron al **Modelo Relacional** predominante y poderoso, pero con algunas limitaciones en la representación de los datos.

- Es un modelo plano, es decir, todo se representa a nivel de tablas, presupone una homogeneidad vertical y horizontal en los datos.
- Las relaciones deben estar en la Primera Forma Normal (INF).
- No incluye explícitamente los aspectos semánticos como parte de la representación de los datos.

- Existen problemas de impedancia, es decir de incompatibilidad que se observa en las diferencias entre el Lenguaje de Consulta que es declarativo y el Lenguaje de Programación el cual es procedural, por lo tanto no embonan, hay "costuras".

Estas limitaciones dieron lugar al desarrollo de los Modelos de Datos, como el Orientado a Objetos y el Lógico, que plantean alternativas de solución a estos problemas.

Capítulo 2

Técnicas de Consulta para Bases de Datos Deductivas

Los **Sistemas de Bases de Datos Deductivas** cuentan con un Lenguaje de Consulta y una Estructura de Almacenamiento diseñados bajo un Modelo Lógico de Datos. Este consiste de una notación matemática para describir los datos y las relaciones, además de una técnica para manipular los datos, la cual hace posible atender las consultas y ver que se cumplan las restricciones de integridad.

En el modelo de Bases de Datos Deductivas la notación matemática que se utiliza para describir los datos, es un Lenguaje de Primer Orden.

La lógica se puede utilizar como un Lenguaje uniforme para expresar Reglas, Hechos, Consultas y Restricciones de Integridad, pues esta ofrece una semántica bien fundada, ya que puede resolver problemas de datos indefinidos y valores nulos. Las Bases de Datos Deductivas no sólo son un estado avanzado de los Sistemas Relacionales, sino que, comparten la propiedad de ser **Declarativos,** es decir se pueden realizar consultas o actualizaciones indicando solamente que se quiere obtener.

2.1 Sintaxis de las Bases de Datos Deductivas

Una Base de Datos Deductiva es un conjunto de Reglas no ordenadas, esta particionada en un conjunto de **Hechos** el cuál es llamado **Base de Datos Extensional** y en un conjunto de todas las **Reglas** llamado **Base de Datos Intensional.**

- **Hechos**

  Se representan por un *predicado* con argumentos constantes.

  La representación extensional significa que la Base de Datos está almacenada por las tuplas verdaderas para el predicado.

- **Reglas**

  Son escritas en una notación del estilo de Prolog, es decir, $p :- q_1, q_2, ..., q_n.$

  En cada uno de los elementos, p representa la cabeza y $q_{i's}$ representan las submetas del cuerpo, donde ambos son fórmulas atómicas.

  En las Bases de Datos Deductivas es deseable que las Reglas sean más numerosas que los Hechos.

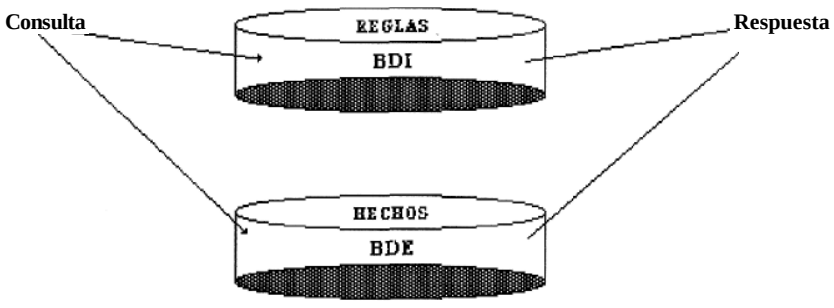


Figura 1. Bases de Datos Extensional e Intensional.

2.1.1 Prolog y Bases de Datos

El Lenguaje **Prolog** en los Sistemas Deductivos es considerado como un lenguaje de consulta adecuado, tomando en cuenta las características de expresividad de los lenguajes de consulta clásicos.

Ventajas del Lenguaje Prolog:

- **Especificación de la Base de Datos Extensional**
  El significado de cada una de las reglas representa el mecanismo para la definición y evaluación de relaciones derivadas.
- **Recursión**
  Tanto los predicados como las reglas pueden ser definidos recursivamente, además aumenta el grado de expresividad.
- **Conocimiento incompleto**
  Se pueden especificar valores nulos tanto en reglas como en hechos del programa cuando una variable toma el valor "_".
- **Orden total en Hechos y Base de reglas**
  Para lograr una ejecución eficiente el programador puede ordenar los hechos y las reglas.
- **Control de Inferencia**
  Utilizando el predicado **cut,** Prolog permite al programador mejorar la eficiencia.
- **Información Negativa**
  Prolog permite expresar información negativa, al introducir un *nil* es más declarativo, más expresivo para el lenguaje.

Desventajas del Lenguaje Prolog:

- La estrategia de evaluación de Prolog es **"depth first" (por profundidad),** la cual puede llevarnos a ciclos infinitos.
- El acceso ineficiente a disco.

Los Sistemas de Prolog procesan programas lógicos recursivos en memoria principal, pero manejan una tupla a la vez. Las Bases de Datos Deductivas representan una alternativa a los problemas de almacenamiento, ya que el mecanismo de deducción del cual están dotadas permite reducir la información almacenada.

La Meta de las Bases de Datos Deductivas es tratar con un superconjunto del Algebra Relacional que incluya soporte para la recursión y mecanismos eficientes de acceso a disco.

2.2 Técnicas de Consulta

Las técnicas se clasifican en:

Objetivo: *Evaluación y Reescritura*
Técnicas de Búsqueda: *"Bottom-Up" y "Top-Down"*

Métodos / Estrategias	EVALUACIÓN	REESCRITURA
BOTTOM UP	Naive	Magic - Sets
		Couting - Sets
	Semi - Naive	Static - Filtering
TOP - DOWN	Query Sub Query	Reducción de Variables Reducción de Constantes

Figura 2. Técnicas de Consulta Deductivas.

2.3 Métodos de Evaluación

Son programas en los cuales dada una consulta y una Base de Datos, se obtiene una respuesta. Los métodos de evaluación se clasifican en **"Bottom_Up",** en esta estrategia las reglas son vistas como *productoras de soluciones y* **"Top_Down"** donde las reglas son vistas como *generadoras de problemas.*

Estrategias Bottom_Up (Ascendente)

En la técnica de búsqueda ascendente el cálculo de tuplas empieza considerando las tuplas que están almacenadas en la Base de Datos y de ahí se derivan todas las tuplas sustituyendo las primeras en el cuerpo de una regla, por lo tanto la respuesta a la consulta es un subconjunto del conjunto de todas las tuplas que satisfacen la meta.

- **Evaluación Ingenua (Naive Evaluation)**

  Es la más simple de las estrategias de evaluación, ésta evalúa la solución al Mínimo Punto Fijo de un predicado.

  Esta evaluación asume un **Predicado y una Base de Datos Extensional**

  - Desventajas

  * *Duplicidad.*- Algunas tuplas son calculadas repetidas veces durante el proceso de iteración hasta alcanzar el mínimo punto fijo.
  * Información Irrelevante.- Algunas tuplas son calculadas sin ser realmente requeridas y se eliminan por la selección final.

- **Evaluación Semi_Ingenua (Semi_Naive_Evaluation)**

  La evaluación Semi_Ingenua es un refinamiento de la Ingenua, usa el mismo enfoque, con excepción que en cada iteración las reglas se aplican únicamente a las tuplas generadas en la iteración previa, éste continúa hasta que no se generan nuevas tuplas.

La Evaluación Semi_Ingenua asume un **Predicado y una Base de Datos Extensional.**

— La Ventaja de la Evaluación Semi_Ingenua sobre la Ingenua es que en cada iteración un término diferencial es usado en lugar del valor total, por lo tanto reduce las tuplas redundantes.

Estrategias Top_Down (Descendentes)

La estrategia de Evaluación Descendente verifica si son necesarias las premisas para que la conclusión se mantenga. Cada meta se considera un problema el cual debe resolverse.

Los algoritmos "Top_Down" deben cumplir las propiedades de *consistencia, completitud y teminación. A* estos algoritmos también se les conoce como **Algoritmos de Optimización,** ya que no se toman en cuenta los hechos que no son útiles en la producción del resultado.

• **Evaluación Consulta_Subconsulta (Query Sub Query** (QSQ))

La estrategia QSQ es general, ya que se aplica a cualquier programa inicial y a la consulta. Es un método Top_Down orientado a conjuntos, ya que procesa las relaciones enteras en lugar de tuplas.

Prolog es diferente de QSQ, ya que éste último actúa bajo "breadth first" el cual está orientado a niveles y genera una condición de salida basada en las nuevas tuplas y en las subconsultas generadas.

— El objetivo de la Evaluación Consulta_Subconsulta (QSQ) es acceder el menor número de hechos, los cuales son necesarios para dar una solución a la consulta.

- El conjunto de hechos generados es mejor que en la estrategia Naive y que en la Semi_Naive, además QSQI tiene los mismos problemas de duplicación que la estrategia Naive.

2.4 Métodos de Reescritura

Estos métodos ejecutan una transformación a un programa. Aquí este programa es escrito bajo el mismo formalismo que el original y el cálculo es más eficiente si se aplica un Método de Evaluación.

Los Métodos de Reescritura se clasifican bajo las estrategias "Bottom_Up" (Conjuntos Mágicos, Conteo y Filtrado Estático) y "Top_Down" (Reducción de Variables y Reducción de Constantes, no analizados en este trabajo).

Definición de Reescritura

Dada una consulta G, un programa P y una transformación τ, el programa obtenido P' por aplicar τ(P), es equivalente a P con respecto a G, ya que tenemos que E(G,P) = E(G,P'), donde E corresponde a un Método de Evaluación.

Los Métodos de Reescritura Lógicos transforman programas ineficientes en eficientes, explotando el argumento instanciado que proporciona la consulta.

2.4.1 Métodos

- **Conjuntos Mágicos (Magic_Sets)**

  Optimiza programas explotando la forma tomada por la consulta. Dada una Regla y una Submeta en el cuerpo de la Regla con algunos argumentos atados, se puede resolver esta Submeta y atar otras variables no atadas en otros argumentos, estas instanciaciones se pueden transferir a otras Submetas en el cuerpo de la Regla y así sucesivamente a otras Submetas.

El Método de Conjuntos Mágicos establece restricciones al programa añadiendo Submetas a las Metas originales.

Las Reglas adicionales restringen las variables del programa para satisfacer otros predicados llamados **"Predicados Mágicos",** las restricciones disminuyen el número de hechos, ya que durante la Evaluación **"Bottom_Up"** las variables toman sólo algunos valores en lugar de todos los posibles.

Este Método consta de un **Predicado,** una **Base de Datos Extensional y una Consulta.**

Las Reglas Mágicas y Complementarias ejecutan la programación "Top_Down" de los valores atados. Las Reglas Modificadas toman las restricciones impuestas por las Reglas Mágicas cuando son utilizadas en el Sistema Adornado.

- **Desarrollo del Método**

1. Dado un sistema de reglas el sistema adornado es obtenido de la siguiente manera:

   Para cada regla y cada adorno del predicado a la izquierda se genera una regla adornada. Cada argumento del predicado se define como distinguido si es constante o si es variable atada en la cabeza de la regla, es definido como *bound (b) y* no distinguido como *free (f).*

2. Las instanciaciones se propagan a través de los predicados de base, la Regla Adornada es obtenida reemplazando cada literal derivada por la versión adornada.
3. Se produce una Regla Mágica para cada ocurrencia del predicado intensional en el lado derecho de cada Regla Adornada.
4. Se genera una Regla Modificada por cada Regla Adornada.
5. Se produce una Regla Complementaria por cada ocurrencia del predicado intensional en el cuerpo de la regla y por cada adorno.

- **Algoritmo**

  - **Paso 1.**
    Las relaciones de la Regla se efectúan explícitamente.
  - **Paso 2.**
    El programa adornado genera las Reglas Mágicas, las cuales simulan la evaluación Bottom_up.
  - **Paso 3.**
    Se modifican las Reglas Adornadas añadiéndoles las Reglas Mágicas generadas en el paso 2.
  - **Paso 4.**
    Se produce una nueva Regla Complementaria por cada predicado de la Base de Datos Intensional y por cada adorno.

- **Ejemplo**

  Para el problema de ancestros definido como:

  ancestro(X, Y) :- padre(X, Y).
  ancestro(X, Y) :- ancestro(X,Z), padre(Z, Y).

  El programa permite calcular los ancestros construyendo un *cono* cuyo vértice es la constante de la consulta requerida.

  El sistema adornado es el siguiente:

  R_1 : *ancestro fb(X,Y) : - padre(X,Y).*
  R_2 : *ancestrofb(X, Y) : - ancestrofb(X, Z), padre(Z, Y).*
  R_3 : *queryf(X) : - ancestro fb(X,oscar).*

- Paso 2.

R_3 : *magic_R_0_ancestrofb(oscar).*

R_2 : *magic_R_2_ancestrofb(Z) : - magic_ancestrofb(Y), padre(Z,Y)..*

- Paso 3.

R_3 : *queryf(X) . - magic_R_0_ancestrofb(oscar), ancestrofb(X,oscar).*

R_2 : *ancestrofb(X,Y) : - magic_R_2_ancestrofb(Z), ancestrofb(X,Z), padre(Z,Y).*

R_1 *: no genera reglas modificadas ya que no existen ocurrencias del predicado intensional*

- Paso 4.

R_3 : *magic_ancestrofb(oscar) : - magic_R_0_ancestrofb(oscar) .*

R_2 : *magic_ancestrofb(Z) : - magic_R_2_ancestrofb(Z).*

A continuación en la figura 3 se presenta la gráfica que ilustra la reescritura antes explicada.

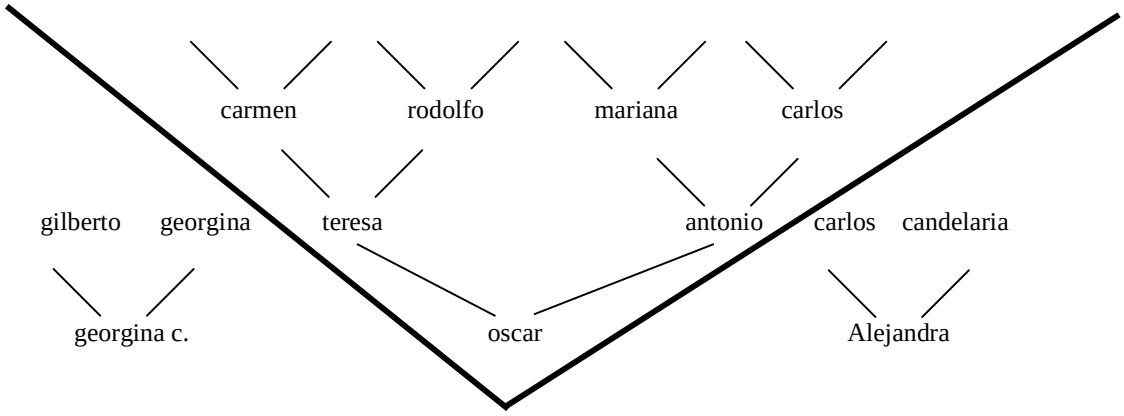


Figura 3. Cono de osear, Conjuntos Mágicos.

- **Método de Conteo (Couting_Sets)**

Este método es un refinamiento del método de conjuntos mágicos. El método incluye el cálculo del conjunto mágico, pero cada elemento es complementado con información adicional dando la distancia de cada nodo a la constante de la consulta, es decir, se establecen niveles dentro del conjunto mágico, el cual restringe el cálculo de tuplas derivadas según la distancia especificada en la meta.

Construcción del Predicado de Conteo

- Cambiar el nombre del predicado magic (mágico), por couting (conteo).
- Aumentar a cada predicado magic un nuevo argumento que expresa la distancia del conjunto mágico a la constante de la meta.

El Método de Conteo es más eficiente que el Método de Conjuntos Mágicos, ya que restringe los niveles de búsqueda.

La duplicación es evitada, ya que las reglas modificadas no contienen las metas previamente resueltas en el cálculo del Conjunto de Conteo. La idea es indexar todos los miembros del conjunto mágico por su distancia a la constante dada.

Al aplicar el Método de Conteo la selección es más precisa durante la evaluación.

Cuando el camino entre dos elementos de la base no es único se crea un problema ya que el concepto de distancia no está bien definido. Si el número de caminos es infinito entonces tenemos una base de datos cíclica. En esta situación el programa reescrito se mantendría contando el mismo arco varias veces.

La base de datos debe ser acíclica y exista al menos una regla linealmente recursiva por cada predicado, para que el método sea consistente, completo y que finalice.

- **Algoritmo**

  - **Paso 1.**
    Se construye el Predicado Mágico.
  - **Paso 2.**
    Al Predicado Mágico se le suma un argumento variable nuevo, que exprese la distancia de cada elemento del Conjunto Mágico a la constante en la consulta.

El programa permite calcular los ancestros que se encuentran dentro del *cono* que ahora sólo se genera hasta el *nivel requerido.*

- **Ejemplo**

  - Paso 1.

    magic_R_0_ancestrofb(oscar).

    counting_R_0(oscar,0).

  - Paso 2.

    magic_R_2_ancestrofb(Z) : - magic_ ancestrofb(Y), padre(Z,Y).
    counting_R_2(Z,I) : - counting(Y,J), padre(Z,Y),I = J + 1.
    queryf(X) . - magic_R_0_ancestrofb(oscar), ancestrofb(X,oscar).
    qf(X,I) : -counting_R_0(oscar,0), ancestrofb_count(X,oscar,I).
    ancestrofb(X,Y) : - padre(X,Y).
    ancestrofb_count(X,Y,I) : - padre(X,Y),integer(I).
    ancestrofb(X,Y) . - magic_R_2_ancestrofb(Z), ancestrofb(X,Z), padre(Z,Y).
    ancestrofb_count(X,Y,I) : - counting_ R_2(Z,I), ancestrofb_count(X,Z,J),padre(Z,Y),I = J – 1.

magic_ancestrofb(oscar) : - magic_R_0_ancestrofb(oscar).
counting(oscar,0) : - counting_R_0(oscar,0).
magic_ancestrofb(Z) : - magic_R_2_ancestrofb(Z).
counting(Z,I) : - counting_R_2(Z,I).

El Sistema de Reglas equivalente obtenido por la transformación de **Conteo** es el siguiente:

counting_R_0(oscar,0).
counting_R_2(Z,I) : - counting(Y,J),padre(Z,Y),I = J + 1.
queryf(X,I) . - counting_R_0(oscar,0), ancestrofb_count(X,oscar,I).
ancestrofb_count(X,Y,I) : - padre(X,Y),integer(I).
ancestrofb_count(X,Y,I) : - counting_R_2(Z,I),ancestrofb_count(X,Z,J),
padre(Z,Y), I = J – 1.
counting(oscar,0) : - counting_R_0(oscar,0).
counting(Z,I) : - counting_R_2(Z,I).

A continuación, en la figura 4 se muestra la optirnización obtenida por el Método de Conteo.

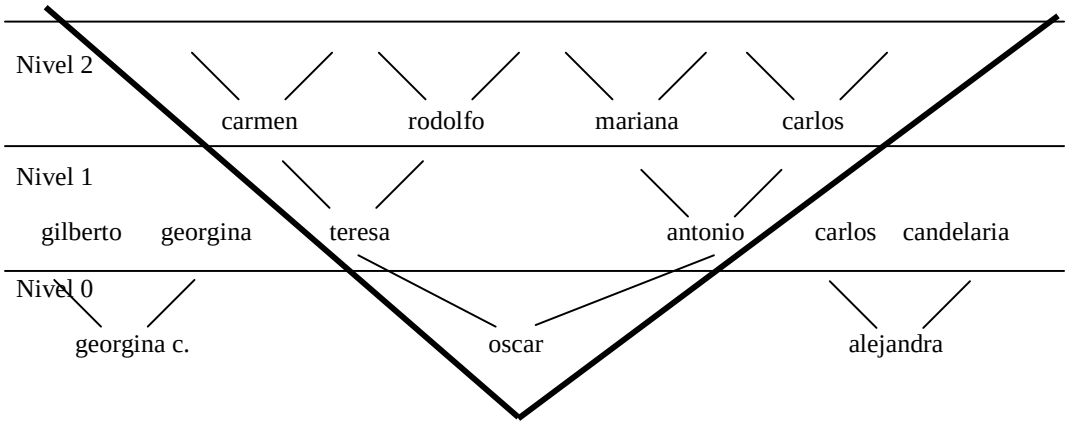


Figura 4. Cono de osear con niveles, Método de Conteo.

- **Método de Filtrado Estático (Static Filtering)**

  Esta estrategia asume un proceso de evaluación "Bottom_Up" a partir de un programa y una meta, construye un grafo llamado **grafo axioma_relación,** para ello se efectúa una sobreescritura de los argumentos para poder diferenciar el mismo argumento, en diferentes ocurrencias del mismo predicado.

  - Es una técnica de optimización que elimina las tuplas irrelevantes del cálculo. La optimización se obtiene mediante la imposición de restricciones llamados **filtros.** El algoritmo debe encontrar los filtros que garanticen las condiciones más restrictivas.

  El **grafo axioma_relación** se puede **optimizar** mediante la propagación de los filtros, desde el puerto de salida del nodo meta hacia todos los nodos relación. La propagación de filtros hacia el resto del grafo es alcanzada transmitiendo las condiciones con respecto a la orientación del grafo tan lejos como sea posible.

  Al entrar las tuplas al nodo de axioma a través de un puerto de entrada se producen tuplas, estas tuplas son transmitidas del puerto de salida del nodo axioma al nodo que sigue.

  El resultado de la optimización del grafo es más eficiente que el del programa original, ya que el segundo argumento en todas las ocurrencias del predicado ancestro está atado.

- Algoritmo
  - Paso 1.
    Se renombran todos los argumentos del Predicado.
  - **Paso 2.**
    Se introduce una condición relacionada con la regla r.
  - **Paso 3.**
    Se propagan los filtros desde la meta hacia todos los nodos de la relación

- **Ejemplo**

Para obtener el Grafo Axioma_Relación se aplican los pasos 1 y 2 del algoritmo.

R_1 : *ancestro(A1, A2) : - padre(P1,P2),* $A1 = P1$, $A2 = P2$.

R_2 : *ancestro*($A1^1$, $A2^1$) : - *padre(P1,P2), ancestro*($A1^2$, $A2^2$), $A1^1 = P1$, $A2^1 = A2^2$, $P2 = Al^2$.

R_3 :*query(Q1) : - ancestro(A1,A2),* $Q1 = Al$, $A2 = oscar$..

A continuación, en la figura 5, se muestra el grafo axioma_relación correspondiente al problema de Ancestros y la consulta ancestro(X,oscar).

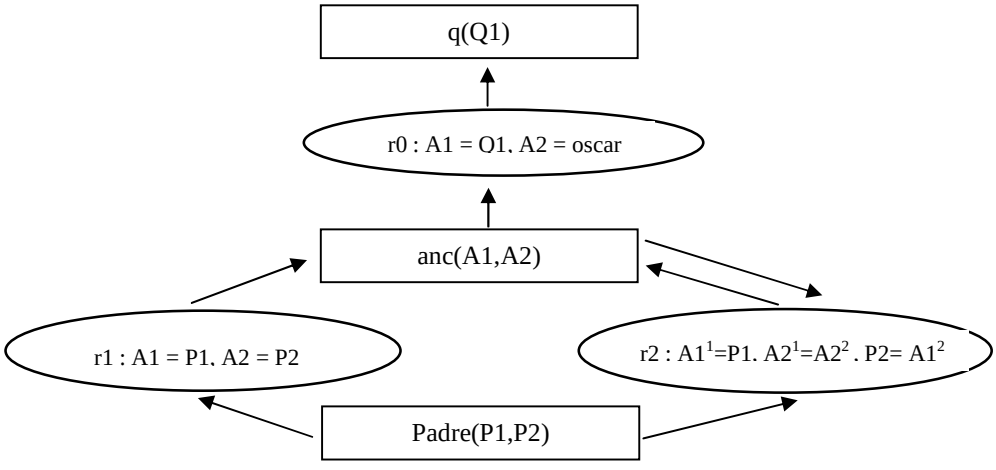


Figura 5. Grafo Axioma_Relación.

El **grafo axioma_relación** consta de dos tipos de nodos, *nodo relación* **p** representado por rectángulos y *nodo de axioma* **r** representado por óvalos, el arco que sale es llamado **puerto de salida** y el que entra **puerto de entrada.**

Capítulo 3

Implantación del Método de Filtrado Estático

Habiendo establecido anteriormente los diversos Métodos de Reescritura, tomemos el Método de Filtrado Estático para definir su implantación.

3.1 Descripción del Método

Este método introduce una optimización a los **Métodos de Evaluación.** En el caso particular de este trabajo se esta optimizando el Método de Evaluación Ingenua.

El método construye un **grafo de axioma_relación** a partir de un programa y una meta.

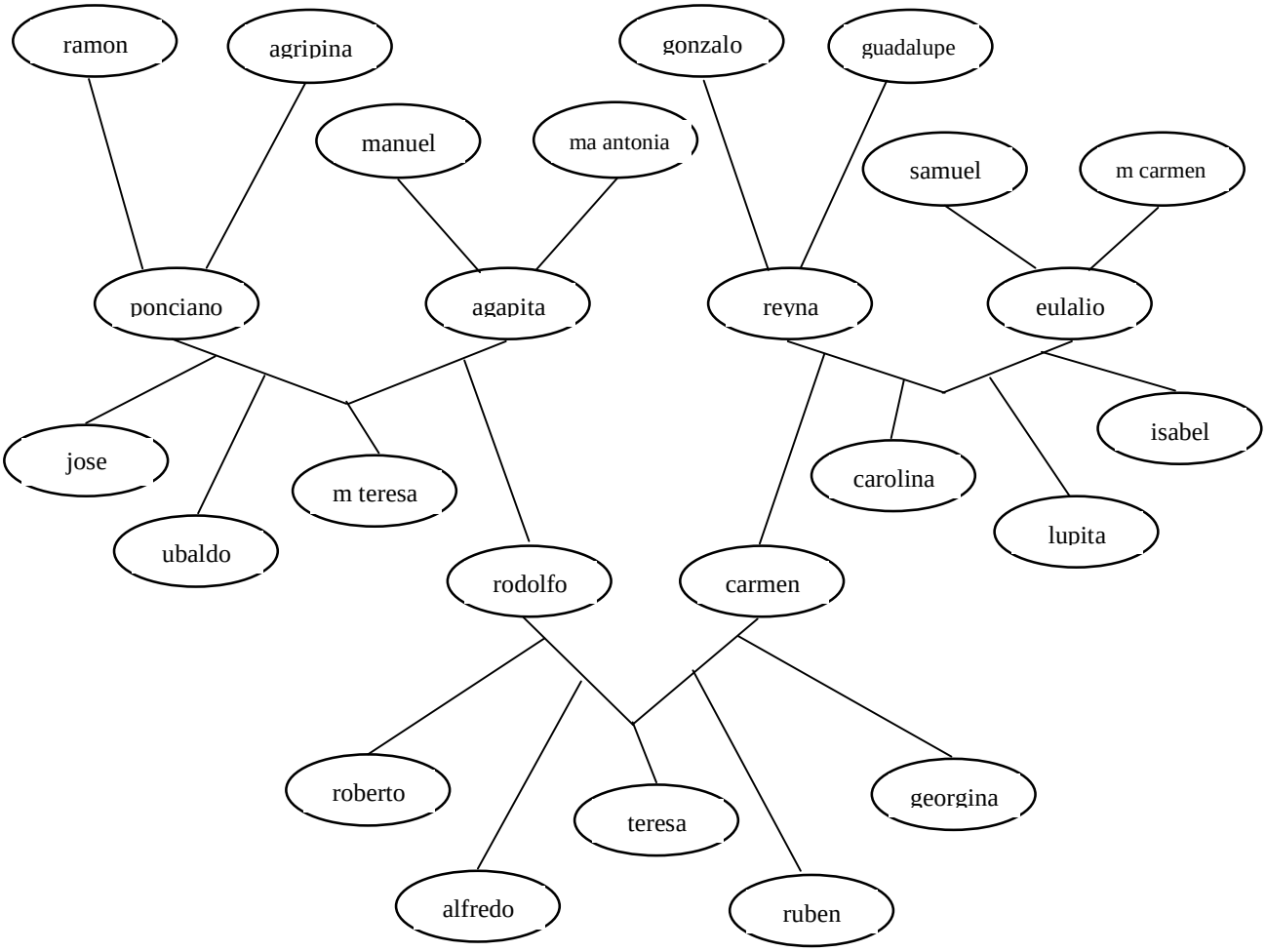
La optimización se logra imponiendo restricciones, las cuales deben ser cumplidas por las tuplas de la solución, es decir, los filtros se propagan desde el puerto de salida del nodo meta hacia todos los nodos relación, quedando las variables atadas.

Las tuplas que no cumplan la condición son cortadas, tan pronto como sea posible, en la primera etapa de su flujo hacia el nodo meta. El método finaliza cuando un nodo de axioma no genera nuevas tuplas.

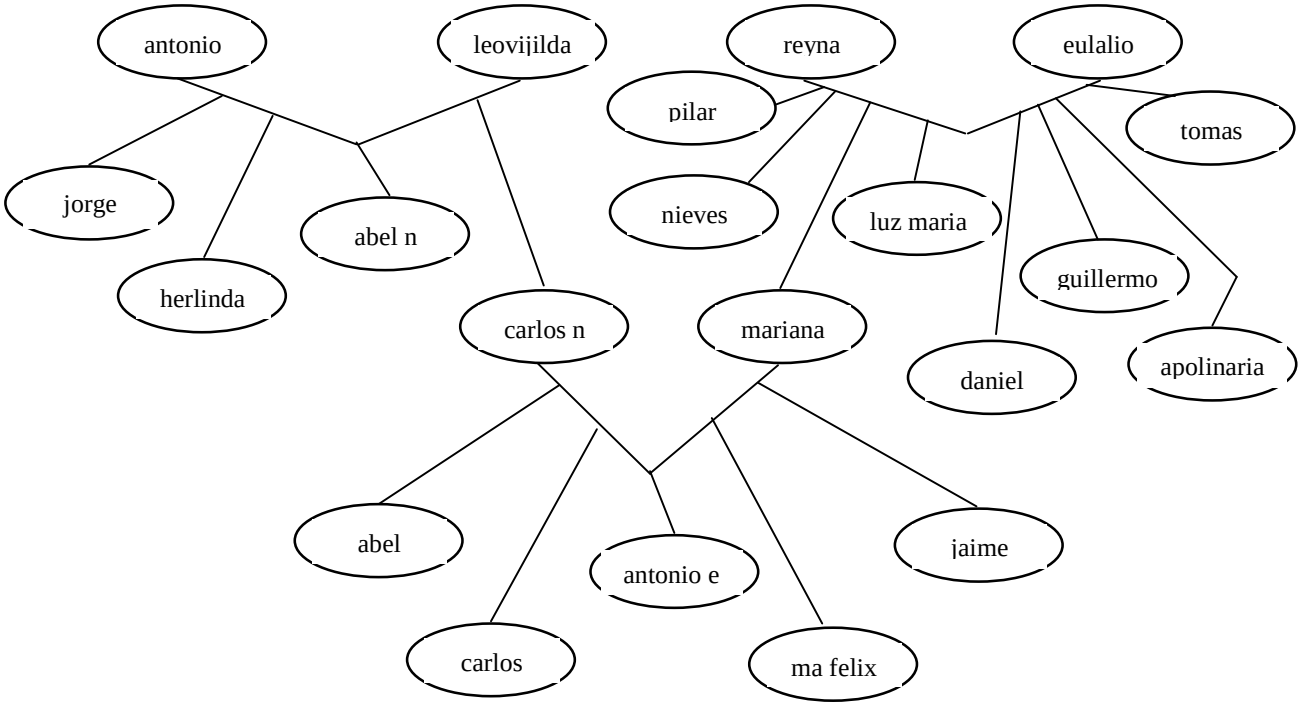
3.2 Descripción de la Base de Datos

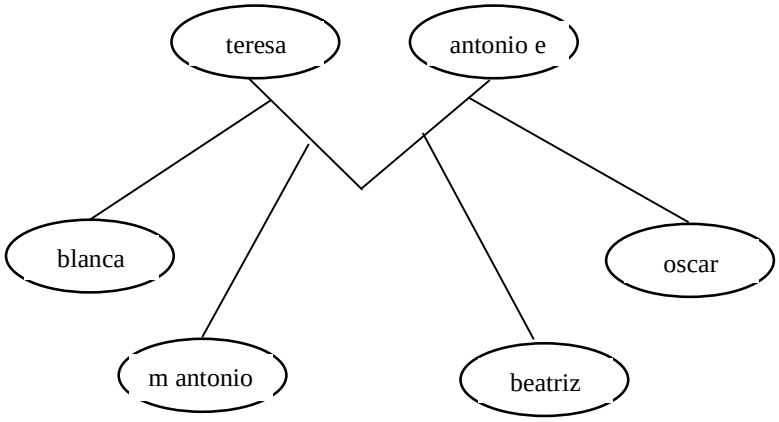
Se utilizó una Base de Datos compuesta por 76 tuplas, con una estructura de dos campos (padre, hijo), ambos son de tipo carácter y longitud 10. La base de datos y la codificación del programa se realizaron en el Sistema de Base de Datos Clipper versión 5.01. El equipo que se utilizó fue una PC 286 de 16 Mhz.

A continuación se muestra el árbol correspondiente a los ancestros, comenzando desde los tatarabuelos.



Continuación del árbol de ancestros.





3.3 Pseudocódigo del Método de Filtrado Estático

Basándose en el Algoritmo del Método de Filtrado Estático presentado en el capítulo 2 se realizó el siguiente pseudocódigo:

- **Paso 1.** *Inicio.*
- **Paso 2.** *Almacena la Base de Datos creada en un arreglo.*
- **Paso 3.** *Selecciona (padre, hijo) inicial.*
- **Paso 4.** *Realiza la búsqueda de los ancestros o descendientes según sea el caso, utilizando el filtro especificado. El filtro corresponde a la constante de la consulta, el cual es propagado a través de ésta.*
- **Paso 5.** *Verifica si la k-ésima tupla ya fue evaluada en una iteración anterior.*
- **Paso 6.** *Almacena en la Base de Datos original las nuevas tuplas encontradas, hasta alcanzar el mínimo punto fijo (cuando las tuplas generadas en la iteración k-1 son iguales a las generadas en la iteración k).*
- **Paso** 7. *Al alcanzar el mínimo punto fijo se imprime el resultado.*
- **Paso 8.** *Fin.*

En el apéndice B se presenta el Código asociado al algoritmo descrito anteriormente y en el apéndice C se desarrollan ejemplos.

Conclusiones

Al finalizar éste trabajo y basándonos en las comparaciones realizadas en el capítulo 2, podemos concluir que los Métodos de Reescritura donde se incluye el Método de Filtrado Estático mejoran considerablemente el desempeño de las Bases de Datos al aplicar un Método de Evaluación de Consultas.

Estos métodos ejecutan una real optimización sobre los programas porque hacen posible que se calculen sólo las tuplas requeridas para la respuesta de la consulta.

Las ventajas asociadas al Método de Filtrado Estático son las siguientes:

- No calcula tuplas irrelevantes.
- No duplica información.
- Aplica una real optimización.
- Disminuye el tiempo de ejecución.
- Disminuye el consumo de memoria.

Al ejecutar una consulta directamente utilizando el Método de Evaluación Ingenua observamos que realiza un número mayor de comparaciones sobre las tuplas en comparación con las calculadas si hubiese sido reescrito aplicando Filtrado Estático. Por lo tanto adiciona tuplas irrelevantes a la base original, siendo estas proporcionales a la base de datos extensional utilizada.

Para un caso particular, con una base original de 76 tuplas al aplicar el Método de Evaluación Ingenua adicionó tuplas, y aplicando el Método de Filtrado Estático adicionó sólo tuplas.

Apéndice A

SQL

En el apéndice A se presentan ejemplos que ilustran la manipulación de los datos utilizando *SQL*.

Ejemplo 1 *Obtener todos los nombres de clientes de la relación Clientes.*

select *nombre*
from *Clientes*

Ejemplo 2 *Obtener a los proveedores cuyo saldo es mayor de 2000.*

select *nombre*
from *Proveedores*
where *saldo > 2000*

Para eliminar duplicados utilizamos la palabra clave **distinct** después de **select,** o bien la palabra **all** para especificar que no se eliminan los duplicados.

SQL usa los conectores lógicos **and, or** y **not**, permite el uso de expresiones aritméticas como operandos de los operadores de comparación.

Una expresión aritmética implica operadores + , —, * y /.

Ejemplo 3 *Obtener el nombre y dirección de todos los proveedores cuyo producto son diskettes5 con un precio mayor de 35.*

select *nombre, dirección*
from *Proveedores*
where *producto = "diskettes5"* **and** *precio > 35*

SQL incluye un operador de comparación **between,** para simplificar cláusulas **where,** análogamente **not between.**

Ejemplo 4 *Obtener los productos de la relación Pedidos cuya cantidad se encuentre entre 20 y 80.*

select *productos*
from *Pedidos*
where *cantidad* **between** *20* **and** *80*

SQL incluye un operador de selección para comparaciones de cadenas de caracteres, el tanto por ciento (%) es igual a cualquier subcadena y subrayado (_) es igual a cualquier carácter. Los caracteres en mayúsculas no son iguales a los caracteres en minúsculas, o viceversa. Los modelos se expresan en *SQL* usando el operador de comparación **like.**

Ejemplo 5 *Obtener los nombres de todos los clientes cuya dirección empiece con Juárez.*

select *nombre*
from *Clientes*
where *dirección* **like** *"Juárez*

El conector in prueba si se es miembro de un conjunto, donde el conjunto es una colección de valores producidos por una cláusula **select,** y **not in** prueba la no pertenencia al conjunto.

Una variable de tupla en *SQL* debe estar asociada con una relación determinada, se definen en la cláusula **from,** colocándola después del nombre de la relación con la cual está asociada, separada por uno o más espacios. Las variables de tupla son muy útiles para comparar dos tuplas de la misma relación.

Ejemplo 6 *Obtener todos los nombres de las personas que hicieron el mismo pedido que Antonio Hernández.*

select *T. nombre*
from *Pedidos S, Pedidos T*
where *S.nombre = "Antonio Hernández"* and
S.nombre = T.producto

En una comparación como **mayor que** no se puede escribir la expresión usando la construcción in. La frase **mayor que algún** se representa en *SQL* por **> some,** también permite las comparaciones **≥ some, < some, ≤ some, = some y ≠ some.** La palabra clave **any** es sinónimo de **some.** La construcción > all corresponde a la frase **mayor que todos,** como en el caso de **some** también permite todas las comparaciones.

Ejemplo 7 *Obtener los nombres de los proveedores, donde el precio del producto sea mayor que el precio de una impresora (impreHPL).*

select *nombre*
from *Proveedores*
where *precio* **> some**
(select *precio*
from *Proveedores*
where *producto = "impreHPL")*

La construcción **exists** devuelve el valor *true* si la subconsulta del argumento no está vacía.

La cláusula **orden by** hace que las tuplas en el resultado dé una consulta en un orden determinado, por omisión *SQL* lista los elementos en orden ascendente, para especificar el tipo de ordenamiento, podemos utilizar **desc** para descentedente o **asc** para ascendente, el orden puede realizarse sobre múltiples atributos.

Ejemplo 8 *Listar en orden alfabético los nombres y direcciones de los proveedores que tienen como producto diskettes de cinco un cuarto (diskettes5).*

```
select    nombre, dirección
from      Proveedores
where     producto = "diskettes5"
orden by  nombre
```

SQL puede calcular funciones en grupos de tuplas usando la cláusula **group by** la cual se usa para formar grupos.

• *SQL* incluye las siguientes funciones de agregación:

- **avg** : Promedio de valores
- **min** : Mínimo valor en la columna
- **max** : Máximo valor en la columna
- **sum** : Suma de los valores
- **count:** Cuenta el número de valores

Las operaciones definidas anteriormente se denominan funciones de agregación porque operan sobre grupos de tuplas. El resultado es un valor único.

Ejemplo 9 *Obtener el saldo promedio de todos los clientes.*

```
select    nombre, avg (saldo)
from      Clientes
group by  nombre
```

La retención de duplicados es importante al calcular un promedio. Pero en algunos casos deben eliminarse antes de calcular una función de agregación, usando la palabra clave **distinct** en la expresión de agregados.

Es útil declarar una condición que se aplica a los grupos más que a las tuplas. La cláusula **having** no se aplica a una única tupla, más bien a cada grupo construido mediante la cláusula **group by.**

Ejemplo 10 *Obtener a todos los clientes cuyo saldo promedio es mayor de 2000.*

select	*nombre,* ***avg****(saldo)*
from	*Clientes*
group by	*nombre*
having ***avg****(saldo)*	*> 2000*

La función de agregación **count** se usa frecuentemente para contar el número de tuplas en una relación. La notación para esto es **count** (*).

Ejemplo 11 *Obtener el número de tuplas en la relación Proveedores.*

select	**count** (*)
from	*Proveedores*

- **Valores Nulos.**

Todas las comparaciones que implican **null** son falsas por definición. La palabra clave especial **null** puede usarse en un predicado para probar si hay o no un valor nulo.

Ejemplo 12 *Obtener todos los clientes, que pertenecen a la relación Pedidos con valores nulos para cantidad.*

select *nombre*
from *Pedidos*
where *cantidad* **is null**

El predicado **is not null** prueba la ausencia de un valor nulo.

A continuación se muestra cómo añadir, eliminar o cambiar información usando *SQL*.

- **Eliminación.**

La Eliminación se expresa casi igual que una consulta, pero solamente se pueden eliminar tuplas completas; no se pueden eliminar valores sólo de atributos determinados. La eliminación de tuplas se expresa como:

delete r
where P

donde r representa una relación y *P* representa un predicado. Las tuplas *t* en r, para las cuales *P(t)* es verdadero, son eliminadas de *r*. Una orden **delete** opera sobre una sola relación. Si se desea eliminar tuplas de varias relaciones, se usa una orden **delete** para cada relación. Podemos tener una cláusula **where** vacía.

Ejemplo 13 *Eliminar todas las tuplas de la relación Pedidos.*

delete *Pedidos*

Aunque se eliminan tuplas solamente de una relación cada vez, podemos hacer referencia a cualquier número de relaciones en un **select-from-where** incorporado en la cláusula **where** de un **delete.**

Ejemplo 14 *Eliminar todos los registros donde el producto sea igual a diskettes de cinco un cuarto (diskettesS).*

delete *Pedidos*
where *producto = "diskettesS"*

Regla Simple: Durante la ejecución de una solicitud de **delete,** solamente se marcan las tuplas que se van a suprimir; no se suprimen realmente. Cuando se termina de procesar la solicitud, esto es, una vez que se hayan marcado las tuplas, entonces se suprimen las tuplas marcadas.

- **Inserción.**

Para insertar datos en una relación, se especifica una tupla que se va a insertar o se escribe una consulta cuyo resultado es un conjunto de tuplas que se van a insertar. Los valores de atributos, deben ser miembros del dominio y deben tener el número correcto de atributos.

Ejemplo 15 *Insertar un cliente nuevo en la relación Clientes*

insert into *Clientes*
values *("Karla Mtnez", "Hidalgo 15", 3200)*

Los valores se especifican en el orden en que se listan los atributos como parte de la sentencia **insert.** También se usa un **select** para especificar un conjunto de tuplas.

- **Actualizaciones.**
  Se puede desear cambiar un valor en una tupla sin cambiar todos los valores, para ello se utiliza la sentencia **update,** es decir, se eligen las tuplas que se van a actualizar usando una consulta.

Ejemplo 16 *Actualizar el precio de una Computadora Acer en la relación de Proveedores.*

update *Proveedores*
set *precio = 10*
where *producto = "CompAcer"*

- **Vistas.**
  Una vista se define en *SQL* usando la orden **create view.** Para definirla debemos dar a la vista un nombre y declarar la consulta que calcula la vista.

La forma de la orden **create view** es:

create view *v* es *expresión de consulta*

donde *expresión de consulta* es cualquier expresión de consulta permitida. El nombre de la vista se representa por *v.* Los nombres de vistas pueden aparecer en cualquier sitio en el que puede aparecer un nombre de relación.

Ejemplo 17 *Crear una vista para obtener un producto y su precio.*

create view *precio-prod* **as**
select *producto, precio*
from *Proveedores*

- **Definición de Datos.**
  El conjunto de relaciones en una base de datos debe ser especificada al sistema por medio de un lenguaje de definición de datos (DDL). El *SQL* DDL, permite la especificación no sólo de un conjunto de relaciones, sino también información sobre cada relación.

- Una relación en SQL se define usando la orden create table:

  create table $r(A_1D_1, A_2D_2, \ldots, A_nD_n]$

  donde r es el nombre de la relación, cada A_i es el nombre de un atributo del esquema de la relación r y D_i es el tipo de datos de los valores en el dominio del atributo A_i. La orden **create table** también incluye opciones para especificar ciertas restricciones de integridad.

  Una relación recién creada está inicialmente vacía.

- Para eleminar una relación de una base de datos en *SQL,* usamos la orden **drop table.** Esta orden elimina toda la información sobre la relación.

  Ejemplo 18 *Eliminar toda la información sobre la relación Clientes.*

  drop table *Clientes*

  Es una acción más drástica que

  delete Clientes

  La última conserva la relación r, pero elimina todas las tuplas. La primera elimina no sólo todas las tuplas sino también el esquema de *r.*

- La orden **alter table** se usa para añadir atributos a una relación existente. A todas las tuplas en la relación se les asigna **null** como valor del nuevo atributo. La forma de la orden es

  alter table ***r*** **add** ***A D***

  donde *r* es el nombre de una relación existente, *A* es el nombre del atributo que se va a añadir y *D* es el dominio del atributo añadido.

Apéndice B

Programa

Descripción del código fuente utilizado.

```
#INCLUDE "BOX.CH"
PAR := {}                              /* Inicia los arreglos de padre y ancestro */
ANC := {}

CLEAR
DATO := SPACE(10)
OPCIÓN := 0
Do WHILE (OPCIÓN <> 3)
  PAR := {}
  ANC := {}
  CLEAR
  DATO := SPACE(10)
  OPCIÓN := 0
  @0,0,23,79 Box B_DOUBLE              /* Realiza la pantalla de presentación */
  @21,1,21,78 Box B_ DOUBLE
  ©5,30,11,49 Box B_ DOUBLE
  SET MESSAGE TO 22 CENTER
  @7,32 PROMPT "ANCESTROS" MESSAGE "EVALÚA LOS ANCESTROS UTILIZANDO EL MÉTODO
  DE FILTRADO ESTÁTICO."
  @8,32 PROMPT "DESCENDIENTES" MESSAGE "EVALÚA LOS DESCENDIENTES UTILIZANDO
  EL MÉTODO DE FILTRADO ESTÁTICO."
  @9,32 PROMPT "SALIDA" MESSAGE "FINALIZA LA SESIÓN DEL PROGRAMA."
```

```
MENU TO OPCION
IF OPCION <> 3
©12,29,16,50 Box B_DOUBLE
@14,30 SAY "FILTRO : " GET DATO              /* Realiza la lectura del filtro */
READ
DATO := LOWER(DATO)
@1,1 CLEAR TO 22,78
@11,27 SAY "PROCESANDO INFORMACIÓN."

USE PADRES
Go TOP
Do WHILE .NOT.EOF()
    AADD(PAR, {PADRE, HIJO})        /* Almacena la base de datos padres en el arreglo PAR */
    SKIP 1
ENDDO
CLOSE DATABASES

BAND := .T.
Do WHILE BAND = .T.
    N := LEN(PAR)
    FOR I := 1 TO N
        P := PAR[I][1]                    /* Selecciona la tupla padre e hijo inicial */
        H := PAR [I] [2]
        FOR J := 1 TO N
            Pl := PAR[J][1]               /* Selecciona la tupla a comparar */
            Hl := PAR[J][2]
            IF OPCIÓN = 1
                IF H = Pl .AND. Hl = DATO    /* Compara si la tupla I-ésima es */
                    AADD(ANC, {P, Hl})       /* ancestro o descendientes según sea */
                ENDIF                        /* el caso, de la tupla J-ésima, además */
            ELSEIF OPCIÓN = 2                /* si el filtro corresponde a la constante /*
                IF H = Pl .AND. P = DATO     /* de la consulta, si ambas se cumplen /* /*
                    AADD(ANC, {P, Hl})       se almacena la tupla ancestro en el /*
                ENDIF                        /* arreglo ANS. /*
            ENDIF
        NEXT
    NEXT
```

```
        M := LEN(ANC)
        Z := LEN(PAR)
        BAND1 := .T.
        FOR K := 1 TO M
            A := 0              /*  Verifica si la K-ésima tupla ya fue evaluada */
            FOR W := 1 TO Z
              IF ANC[K][l] = PAR[W][1] .AND. ANC[K][2] = PAR[W][2]
                 A := 1
               ENDIF
            NEXT
            IF A = 0    /*Si la K-ésima tupla no fue encontrada la almacena en el arreglo PAR*/
         AADD (PAR, {ANC[K][l], ANC[K][2]})
         BAND1 := .F.
         ENDIF
         NEXT
        IF BAND1 = .T.
           BAND := .F.
         ENDIF
        ANC := {}
ENDDO

CLEAR
@0,0,23,79 Box B_ DOUBLE            /* Imprime pantalla de resultado */
@2,24 SAY "PADRE"
@2,44 SAY "HIJO"
U := LEN(PAR)
Y := 3
FOR V := 1 TO U
    IF Y > 17
        @22,45 SAY "PRESIONE ENTER PARA CONTINUAR."
        INKEY(0)
        @3,23 CLEAR TO 18,70
        Y := 3
    ENDIF
```

```
        IF OPCIOn = 1
            IF DATO = PAR[V][2]
                @ Y,23 SAY PAR[V][1]          /* Imprime el resultado de la consulta ancestro */
                @ Y,43 SAY PAR.[V][2]
                Y := Y+l
            ENDIF
         ELSEIF OPCIÓN = 2
            IF DATO = PAR[V][1]
                @ Y,23 SAY PAR[V][1]        /* Imprime el resultado de la consulta descendientes */
                @ Y ,43 SAY PAR[V][2]
                Y := Y+l
            ENDIF
         ENDIF
    NEXT
    @22,45 SAY "PRESIONE ENTER PARA CONTINUAR."
     INKEY(0)
     ENDIF
ENDDO
CLEAR
RETURN .T.
```

Apéndice C

Ejemplos

Demostración de la ejecución de las consultas:

- Método de Evaluación Ingenua (12 tuplas).
  - *ancestro (A, oscar).*
  - *descendientes (mariana,D).*
- Método de Filtrado Estático (12 tuplas).
  - *ancestro (A, oscar).*
  - *descendientes (mariana, D).*
- Método de Filtrado Estático (76 tuplas).
  - *ancestro (A, blanca).*
  - *descendientes (ramón,D).*

Cabe hacer notar en el desarrollo de los ejemplos, la optimización que introduce el Método de Filtrado Estático, pues es significativa la disminución de tuplas calculadas.

Ejemplo 1 *Aplicación del Método de Evaluación Ingenua para el cálculo de las tuplas de la consulta ancestro (A, oscar).*

— *Pantalla de Menú Principal.*

— *Pantalla de Resultado.*

TUPLAS GENERADAS POR EVALUACION INGENUA.

rodolfo	teresa	carmen	blanca
carmen	teresa	carmen	m antonio
carlos	antonio	carmen	beatriz
mariana	antonio	carmen	oscar
antonio	blanca	carlos	blanca
teresa	blanca	carlos	m antonio
antonio	m antonio	carlos	beatriz
teresa	m antonio	carlos	oscar
antonio	beatriz	mariana	blanca
teresa	beatriz	mariana	m antonio
antonio	oscar	mariana	beatriz
teresa	oscar	mariana	oscar
rodolfo	blanca		
rodolfo	m antonio		
rodolfo	beatriz		
rodolfo	oscar		

RESULTADO

PADRE	HIJO
antonio	oscar
teresa	oscar
rodolfo	oscar
carmen	oscar
carlos	oscar
mariana	oscar

Presione ENTER para continuar.

Ejemplo 2 *Aplicación del Método de Filtrado Estático para el cálculo de las tuplas de la consulta ancestro (A, oscar).*

~ *Pantalla de Menú Principal y lectura de Filtro.*

— *Pantalla de Resultado.*

TUPLAS GENERADAS		PADRE	HIJO
rodolfo	teresa	antonio	oscar
carmen	teresa	teresa	oscar
carlos	antonio	Rodolfo	oscar
mariana	antonio	carmen	oscar
antonio	blanca	carlos	oscar
teresa	blanca	mariana	oscar
antonio	m antonio		
teresa	m antonio		
antonio	beatriz		
teresa	beatriz		
antonio	oscar		
teresa	oscar		
rodolfo	oscar		
carmen	oscar		
carlos	oscar		
mariana	oscar		

Presione ENTER para continuar.

Ejemplo 3 *Aplicación del Método de Evaluación Ingenua para el cálculo de las tuplas de la consulta descendientes(mariana,D).*

— *Pantalla de Menú Principal.*

— *Pantalla de Resultado.*

TUPLAS GENERADAS POR EVALUACION INGENUA.

rodolfo	teresa	carmen	blanca
carmen	teresa	carmen	m antonio
carlos	antonio	carmen	beatriz
mariana	antonio	carmen	oscar
antonio	blanca	carlos	blanca
teresa	blanca	carlos	m antonio
antonio	m antonio	carlos	beatriz
teresa	m antonio	carlos	oscar
antonio	beatriz	mariana	blanca
teresa	beatriz	mariana	m antonio
antonio	oscar	mariana	beatriz
teresa	oscar	mariana	oscar
rodolfo	blanca		
rodolfo	m antonio		
rodolfo	beatriz		
rodolfo	oscar		

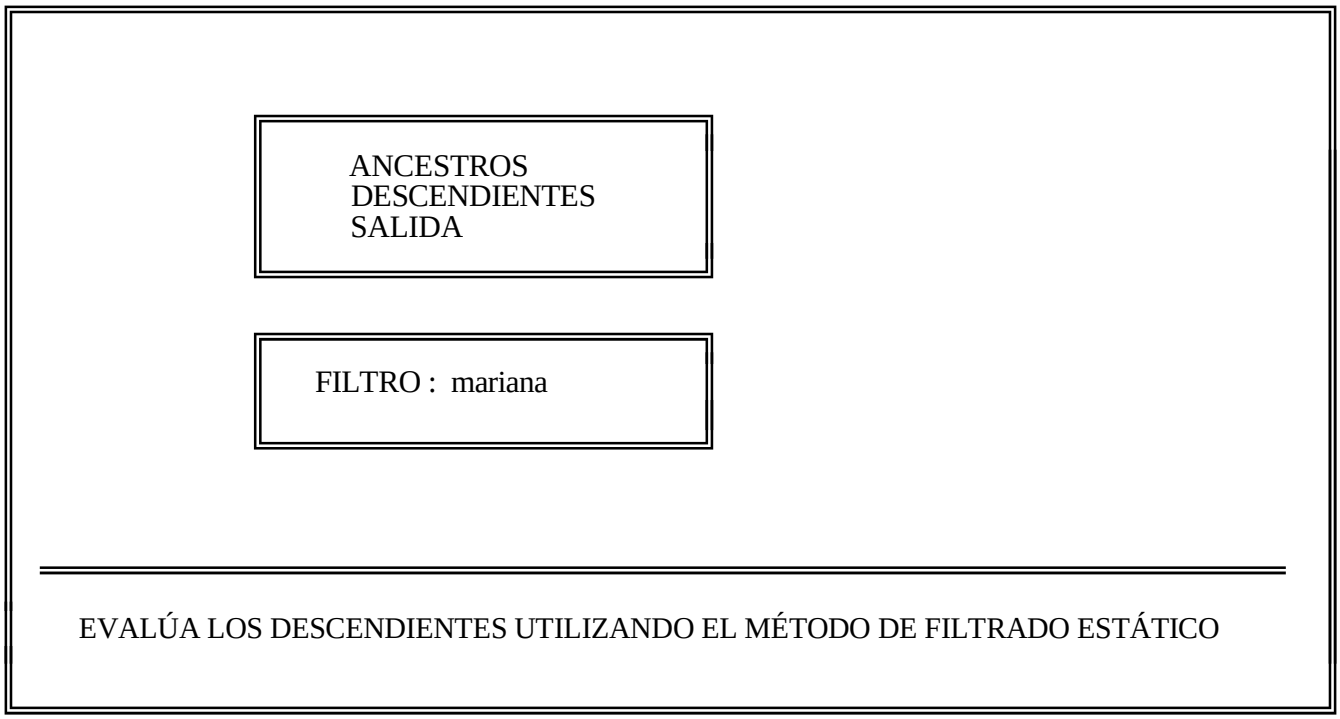
RESULTADO.

PADRE	HIJO
mariana	antonio
mariana	blanca
mariana	m antonio
mariana	beatriz
mariana	oscar

Presione ENTER para continuar.

Ejemplo 4 *Aplicación del Método de Filtrado Estático para el cálculo de las tuplas de la consulta descendientes (mariana,D).*

— *Pantalla de Menú Principal y lectura de Filtro.*



— *Pantalla de Resultado.*

TUPLAS GENERADAS		PADRE	HIJO
rodolfo	teresa	mariana	antonio
carmen	teresa	mariana	blanca
carlos	antonio	mariana	m antonio
mariana	antonio	mariana	beatriz
antonio	blanca	mariana	oscar
teresa	blanca		
antonio	m antonio		
teresa	m antonio		
antonio	beatriz		
teresa	beatriz		
antonio	oscar		
teresa	oscar		
mariana	blanca		
mariana	m antonio		
mariana	beatriz		
mariana	oscar		

Presione ENTER para continuar.

Ejemplo 5 *Aplicación del Método de Filtrado Estático para el cálculo de las tuplas de la consulta ancestro (A, blanca).*

—Pantalla de Menú Principal y lectura de Filtro.

— *Pantalla de Resultado.*

PADRE	HIJO
antonio e	blanca
teresa	blanca
carlos n	blanca
mariana	blanca
rodolfo	blanca
carmen	blanca
ponciano	blanca
agapita	blanca
eulalio	blanca
reyna	blanca
antonio	blanca
leovijilda	blanca
prospero	blanca
romualda	blanca
ramón	blanca

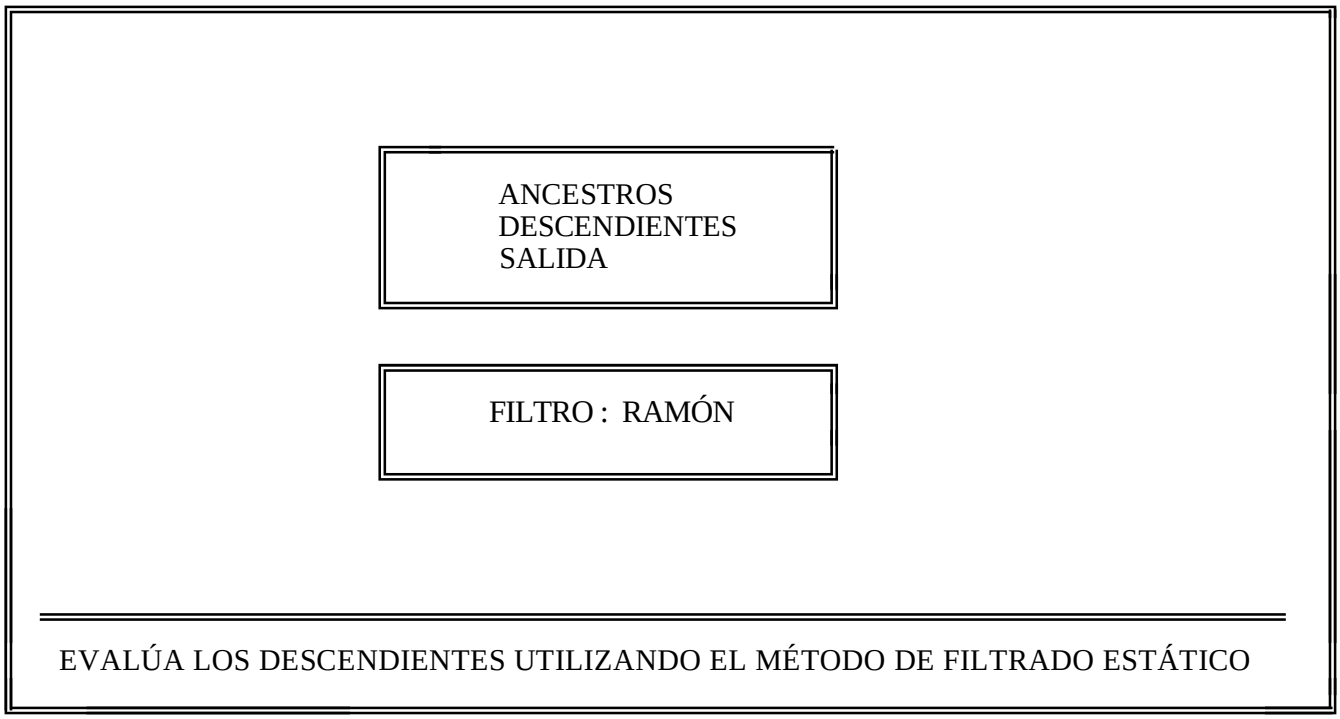
Presione ENTER para continuar.

PADRE	HIJO
agripina	blanca
manuel	blanca
ma antonia	blanca
gonzalo	blanca
guadalupe	blanca
samuel	blanca
m carmen	blanca

Presione ENTER para continuar.

Ejemplo 6 *Aplicación del Método de Filtrado Estático para el cálculo de las tuplas de la consulta descendientes (ramón, D).*

— *Pantalla de Menú Principal y lectura de Filtro.*



— *Pantalla de Resultado.*

PADRE	HIJO
ramón	ponciano
ramón	jóse
ramón	ubaldo
ramón	m teres a
ramón	rodolfo
ramón	roberto
ramón	alfredo
ramón	teresa
ramón	ruben
ramón	georgina
ramón	blanca
ramón	m antonio
ramón	beatriz
ramón	oscar

Presione ENTER para continuar.