CENTRO DE INVESTIGACIONES Y ESTUDIOS AVANZADOS

DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION COMPUTACION

"IMPLANTACION DEL LENGUAJE DE CONSULTA QBE"

Tesis que presenta el Ing. Francisco Javier Juárez Yáñez para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERIA ELECTRICA. Trabajo de Tesis dirigido por el Dr. Renato Barrera Rivera.

Becario de COSNET

México D. F., Octubre de 1987.

R E S U M E N

Query-by-Example (QBE) es un Lenguage de Alto Nivel diseñado para la manipulación y definición de Bases de Datos Relacionales. Provee un estilo muy peculiar de realizar una solicitud, actualizar, definir y controlar una base de datos. Por la filosofía de QBE se requiere que el usuario conozca muy poco acerca de programación y bases de datos para poder explotar el lenguaje.

Este trabajo de tesis trata de una implantación del lenguaje Query-by-Example para una microcomputadora PC-IBM o compatible. La implantación se hace usando los lenguajes de programación 'C' y 'Clipper'. El trabajo solo contempla la parte de QBE que maneja la base de datos para la elaboración de consultas.

El programa de computo del sistema consta de tres procesos, donde cada uno corresponde a una etapa de la implantación. Estas etapas son:

- Etapa Gráfica: En ésta se le brinda al usuario la posibilidad de generar su solicitud a la base de datos, usando la notación dada para QBE. Tambien aquí se pasa la solicitud hecha por el usuario, a una equivalente en notación lineal, y ésta es la que entra a la siguiente etapa.

- Etapa de Generación del Arbol Sintático: Se genera a partir de la solicitud, en notación lineal, realizada por el usuario un árbol binario de sintáxis, donde cada nodo del mismo hace referencia a una Operación del Algebra Relacional. El árbol en su conjunto contendrá una solicitud equivalente a la que se tuvo como entrada de la etapa, pero en notación de operaciones algebraicas.

- Etapa de Recorrido del Arbol: Aquí se realiza el recorrido del árbol generado en la etapa anterior. En cada 'visita' de un nodo del árbol, se efectua la operación que se especifica en él mismo sobre alguna relación de la base, o de las que se generaron en visitas anteriores a otros nodos.

Los dos primeros capítulos de la tesis hablan sobre bases de datos y la definición de QBE. Los últimos tres tratan sobre los detalles de la implantación de las tres etapas citadas. Se tienen cinco apéndices, donde el contenido de cada uno es: del primero el de una base de datos ficticia y la solución de consultas realizadas sobre ésta, que se dan para ilustrar la forma de operación del sistema; del segundo, una lista de las decisiones de diseño adoptadas para esta implantación de QBE; del tercero, la definición de la sintaxis original de QBE; del cuarto, una lista de los requerimientos mínimos de operación del sistema de esta implantacion; y del quinto, ejemplos de corridas del sistema.

I N D I C E

CAPITULO I

BASES DE DATOS

La creciente necesidad de manejar eficientemente grandes cantidades de información por parte de cualquier persona, y el gran problema que representa el hacerlo, han provocado que se busque la automatización de estas tareas mediante computadoras. Con esta idea aparecieron las denominadas 'Bases de Datos', que no son otra cosa que 'Un Sistema de grabado y manejo de información basado en computadoras'.

El uso de la base de datos presenta para el usuario, entre otras cosas, las siguientes ventajas:

- Puede reducir y controlar la Redundancia en la información almacenada.

- Es posible eliminar la Inconsistencia en la información.

- Los datos de la base pueden ser compartidos sin la necesidad de tener más de una copia de la información.

- Se pueden seguir estandares de almacenamiento de la información, para permitir el intercambio de datos.

- Se pueden aplicar restricciones de seguridad para la consulta y actualizacion de la información.

- Puede ser mantenida la Integridad de la información.

Dado que la gran mayoria de quienes tienen necesidad de manejar una base de datos son personas sin grandes conocimientos en programación de computadoras, se ha buscado que el realizar esta tarea les resulte lo más fácil y eficiente posible. Es por ésto que se ha tratado de contar con lenguajes de manipulación y definición de Bases de Datos que permitan a 'cualquier usuario' realizar consultas y actualizaciones de esta sin que esto represente para él un gran problema.

"Query By Example" (QBE) es un lenguaje de consulta de datos, y su aplicación está dirigida a la Manipulación y Definición de Bases de Datos Relacionales. Este presenta las siguientes características:

- Está dirigido al Manejo y Definición de Bases de Datos Relacionales.

- Resulta ser bastante fácil de aprender por cualquier usuario final, aun cuando este conozca poco (o nada) sobre las computadoras y sus lenguajes de programación. Se pretende que

el usuario sea capaz de aprenderlo a manejar perfectamente y consecuentemente de explotarlo en una forma bastante productiva tras pocas horas de aprendizaje.

- Resulta ser bastante agradable para su uso por la 'animación' que tiene el Sistema en el cual se implementa, .

- La definición del lenguaje es idéntica a la de un Lenguaje Gráfico, como los que se usan para las matemáticas, ya que hace uso de 'Esqueletos de Tablas', Expresiones definidas en los encabezados de las columnas de las tablas, y de Comandos dados en cada columna de las tablas.

Con la llegada de este tipo de lenguajes gráficos se pretende:

- Difundir el uso de la computadora, esto es, que no solo personas con conocimientos especializados puedan hacer uso de esta.

- Hacer más accesibles las bases de datos al usuario no-técnico.

- Hacer más dinámica su aplicación.

CAPITULO II

Q U E R Y B Y E X A M P L E
("QBE")

Query by Example es un Lenguje de alto nivel para el manejo de Bases de Datos relacionales. Presenta la particularidad de ser un 'Lenguaje Gráfico' y de que el usuario realiza el manejo de la base a través de operaciones expresadas en una notación bidimensional.

La Implantación de QBE, que es tema de este trabajo, presenta pequeñas diferencias en relación con la definicion dada por M. M. Zloof, pero no por esto pierde sus grandes caracteristicas, que son:

- No requiere que el usuario conozca mucho acerca de Lenguajes de Programación y Bases de Datos, y son muy pocos los conceptos acerca de QBE que debe de aprender para entenderlo y manejarlo totalemente.

- La sintaxis del lenguaje es bastante simple, ya que con unos pocos símbolos se puede expresar un gran conjunto de solicitudes a la base de datos.

- El usuario requiere de muy pocas horas de estudio del lenguaje para poder realizar solicitudes bastantes complejas.

Asi, la descripción de lo que es y de lo que se puede hacer con QBE, la haré basandome en la implementación que adopté.

El nombre de 'Query by Example' (Solicitud mediante ejemplo), deriva su nombre del hecho de que la solicitud se hace dando un ejemplo de la forma en que se debe manipular la base de datos, y de cómo se espera el resultado de tal solicitud. Esta consulta la hace el usuario poniendo su 'ejemplo' en un 'Esqueleto' de una tabla en pantalla (inicialmente en blanco). Asi entonces, si se cuenta con una base de datos que maneja información de determinados entes de una Almacen de Ventas, mediante las siguientes relaciones:

- Empleado (Nombre, Salario, Departamento)
- Ventas (Departamento, Articulo)
- Surtido (Articulo, Proveedor)
- Tipo (Articulo, Color, Tamaño)

Si el usuario desea hacer la solicitud (a la relación correspondiente): 'Dame todos los artículos que vengan en color "Verde"', usando la notación de QBE se tendria lo siguiente:

TIPO	ARTI	COLOR	TAMAÑO
	I. E. GOMA	C. VERDE	

En donde la "I." indica que se desea que se imprima el 'Nombre de los Articulos', por ejemplo 'Goma', de aquellos que vengan en color 'Verde'. El comando "E" indica que se está definiendo un ejemplo de un artículo que se puede tener en la impresión (como puede ser una 'Goma'), y el comando "C" define que esos articulos deben ser de color verde.

La nomenclatura de Comandos y Operadores que requiere QBE para representar sus solicitudes, es bastante reducida y simple, siendo ésta:

i).- E. <ele-eje> - Muestra un "Elemento Ejemplo" de lo que se espera como resultado de la solicitud. Tambien sirve para encadenar 2 campos de diferentes relaciones en solicitudes complejas. Este elemento no es necesario que aparezca como contenido del campo, en el que se pone, de algún registro en la relación.

ii).- C. <ele-cons> - Indica un "Elemento Constante", y se espera que éste aparezca como contenido del campo en el que aparece, de algun(os) registro(s) de la relación.

iii).- I. - Indica que se desea imprimir como resultado (para aquellos registros generados por la solicitud) el contenido del campo en el que aparece este operador.

iv).- A. - Define que la relación en la que aparece este operador se fraccionará en grupos, en donde cada uno de estos deberá tener la misma información en el campo en donde está el operador.

v).- Todo. - Indica que la relacion en la cual aparece, deberá de tratarse como una unidad (grupo), a la que se le podrá aplicar una Función Prefabricada (explicadas a continuación), o a la que podrá considerarse como un conjunto del que se desea saber si contiene a (o está contenido en) algún otro.

vi).- SUM., CNT., PRM., MAX., MIN. - Funciones Prefabricadas aritméticas, que se aplican sobre relaciones que fueron agrupadas, y que generan un resultado numérico.

vii).- =, >, <, ~ - Operadores Relacionales, usados para definir operaciones de Seleccion de elementos constantes y de Encadenamiento entre relaciones. Estas operaciones pueden ser aplicadas a grupos.

viii).- "¬" - Operador de Negación se usa para negar toda la relación resultante de aplicar a la base las operaciones especificadas en la tabla (en la que aparece el operador). Se especifica en la columna del nombre, debajo de este, de la relación asociada a la tabla.

ix).- Esqueleto - Usado para definir una Relacion de la Base de Datos , asi como los campos de ésta (la relación) a usar en la solicitud . Los nombres de la relación y de los campos se especifican en el esqueleto, en los siguientes lugares:

RELACION	CAMPO 1	CAMPO 2	CAMPO 3

En donde:

Relación.- Es el nombre de la relación, asociada al esqueleto de la tabla, sobre la que el usuario desea que se realicen ciertas operaciones que definirá en tal tabla.
Campo 1, Campo 2, Campo 3.- Son campos de la relación especificada, sobre los que se operará.

Debido a que la implantación del Lenguaje QBE que se realizó en este trabajo, esta dirigida a usuarios que habla la lengua castellana, la nomenclatura de aquellos comandos que terminan con un '.' difiere de la definición original hecha por Zloof para éstos. Esto se hizo con el fin de que la nueva nomenclatura para estos comandos, exprese por si misma, de acuerdo al castellano, su significado.

A continuación se muestran diferentes tipos de recuperaciones disponibles en QBE. En cada uno se dá la solicitud formulada en castellano, luego su especificación en el lenguaje gráfico QBE y finalmente su represetación lineal (ver apéndice I).

1).- Recuperación Simple.- 'Da todos los colores de los diferentes articulos'. La forma en que se haría esta solicitud en QBE es:

TIPO	ARTI	COLOR	TAMAÑO
		I. E. BLANCO	

Figura 1

Notación Lineal:

Tipo (Color: I. E. Blanco)

Alternativamente, 'E. BLANCO' pudo haber sido omitido, sin alterarse el resultado de la solicitud.

2).- Recuperación Simple con multiples impresiones.- 'Da todos los artículos disponibles en el almacén y sus caracteristicas':

TIPO	ARTI	COLOR	TAMAÑO
	I.	I.	I.

Figura 2

Notación Lineal:

Tipo (Arti: I., Color: I., Tamaño: I.)

Alternativamente algun(os) elementos ejemplo pudieron haber sido usados.

3).- Recuperación Restringida.- Considérese la siguiente solicitud: 'Imprime los nombres de los empleados que trabajan en el Departamento de "JUGUETERIA", y ganen más de "$10000"'.

EMP	NOMBRE	SAL	DEPT
	I.	C. > 10000	C. JUGUETERIA

Figura 3

Notación Lineal:

Emp (Nombre: I., Sal: C. > 10000, Dept: C. Jugueteria)

Como se apuntó, los Operadores Relacionales con que se cuentan son: =, <, > y . Si no se especifica algun operador, se presupone el de igualdad (=). El operador '¬' nos indica desigualdad.

4).- Recuperación Restringida usando encadenamientos.- 'Imprime el nombre de todos los articulos que vengan en color "VERDE", y que sean vendidos por el departemento de "JUGUETERIA"':

TIPO	ARTI	COLOR	TAMAÑO
	I. E. CLAVO	C. VERDE	

VENTAS	DEPT	ARTI
	C. JUGUETERIA	E. CLAVO

Figura 4

Notación Lineal:

Tipo
Type (Arti: I. E. Clavo, Color: C. Verde)
Ventas (Dept: C. Jugueteria, Arti: E. Clavo)

En éste caso es necesario el uso de dos Tablas, una para cada relación empleada en la solicitud. Aquí el mismo elemento ejemplo debe ser usado en ambas tablas, indicando que si un artículo ejemplo, tal como "CLAVO" es "VERDE", éste debe ser vendido en el departamento de "JUGUETERIA".

5).- Elementos Ejemplo encadenados en la misma tabla.- 'Encuentra los nombres y salarios de los empleados que ganan más que "SANCHEZ"':

EMP	NOMBRE	SAL	
	I.	I. E. > S1	

EMP	NOMBRE	SAL	
	C. SANCHEZ	E. S1	

Figura 5

Notación Lineal:

Emp (Nombre: I., Sal: I. E. > S1)
Emp (Nombre: C. Sanchez, Sal: E. S1)

6).- Recuperación usando una Negación.- 'Imprime los departamentos que no venden algun artículo provisto por "PARKER"':

VENTAS	DEPT	ARTI
	I.	E. TINTA

SURTIDO	ARTI	PROVEEDOR
	E. TINTA	C. PARKER

Figura 6

Notación Lineal:

Ventas (Dept: I., Arti: E. Tinta)
¬ Surtido (Arti: E. Tinta, Proveedor: C. Parker)

En este caso el operador de negación '¬' se aplica a toda la expresión de la solicitud en la tabla SURTIDO. Para este tipo de solicitud lo que hará el sistema es: determinar todos aquellos articulos, como puede ser "TINTA", provistos por "PARKER", y sólamente imprimirá el nombre de aquellos Departamentos en los que no sea el caso de que algunos de sus artículos es de los que se determinaron previamente.

7).- Recuperación usando una Negación.- 'Imprime los nombres de los departamentos que venden algún artículo no provisto por "PARKER"':

VENTAS	DEPT	ARTI
	I.	E. TINTA

SURTIDO	ARTI	PROVEEDOR
	E. TINTA	C. ^ PARKER

Figura 7

Notación Lineal:

Ventas (Dept: I., Arti: E. Tinta)
Surtido (Arti: E. Tinta, Proveedor: C. ^ Parker)

8).- Recuperación de Multiples Tablas usando Tabla de Resultados.- 'Recupera cada uno de los departamentos con sus correspondientes proveedores':

SURTIDO	ARTI	PROVEEDOR
	E. TINTA	E. IBM

VENTAS	DEPT	ARTI
	E. DISCO	E. TINTA

TABLA RESULTA	COSAS	XXX
	I. E. DISCO	I. E. IBM

Figura 8

Notacion Lineal:

Surtido (Arti: E. Tinta, Proveedor: E. IBM)
Ventas (Dept: E. Disco, Arti: E. Tinta)
Tabla Resulta (Cosas: I. E. Disco, XXX: I. E. IBM)

Por las características de funcionamiento de QBE, el operador de impresión "I." solo puede aparecer en una tabla. Así cuando se tiene una solicitud cuyo resultado es el contenido de campos que se encuentran en relaciones diferentes, es necesario crear una tabla extra que no estará asociada con ninguna relación de la base. En esta tabla se definen los campos a recuperar mediante encadenamientos de elementos ejemplos especificados en esta tabla, con los campos de las relaciones de interés. Los encabezados de esta tabla extra pueden o no ir especificados, y en caso de que se pongan, pueden ser cualquiera que seleccione el usuario. En la implantación que se realizó, a la tabla extra generada para este tipo de solicitudes se le denomina "Tabla de Resultados", y llevará como encabezado de nombre de la relación la frase: "TABLA RESULTA".

9).- Recuperación usando operadores AND.- 'Dame el nombre de aquellos empleados que trabajan en el departamento de "JUGUETERIA", y ganan más de "$10000"':

EMP	NOMBRE	SAL	DEPT
	I.	C. > 10000	C. JUGUETERIA

Figura 9

Notación Lineal:

Emp (Nombre: I., Sal: C. > 10000, Dept: C. Jugueteria)

En este caso se especifican dos elementos constantes en respectivos campos de una misma relación. El operador lógico AND está implícito en la solicitud realizada.

10).- Recuperación usando operadores AND, OR y una Caja de Condición.- Otro tipo de solicitud que emplea al operador AND y lo utiliza explícitamente es: 'Dame el "NOMBRE" de aquellos empleados cuyo "Salario es mayor de $10000, menor de $15000, pero no es igual a $13000"':

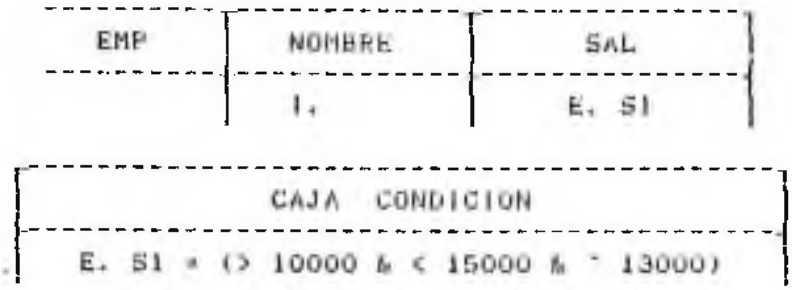


Figura 10

Notación Lineal:

```
Emp (Nombre: I., Sal: E. S1)
Caja Condicion (E. S1 = (> 10000 & < 15000 ~ 13000))
```

En esta solicitud se observa un nuevo elemento de QBE que es la 'Caja de Condición', siendo ésta una tabla que tiene el encabezado "CAJA CONDICION", y que sirve para que el usuario exprese en ella una condición que sería difícil de poner en las tablas. Dentro del lenguaje el operador lógico AND se representa mediante el símbolo '&'. En este caso se están asociando varias condiciones a un determinado campo de una relación.

El Operador OR en QBE se representa mediante el símbolo '|', así para la solicitud 'Recupera el "NOMBRE" de aquellos empleados cuyo "Salario sea de $10000 o $13000 o $16000"', se tiene:

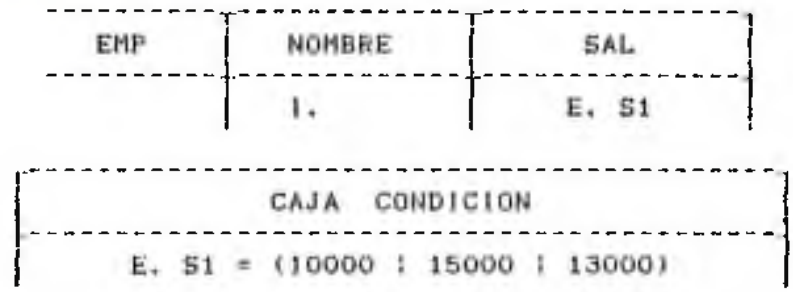


Figura 11

Notación Lineal:

```
Emp (Nombre: I., Sal: E. S1)
Caja Condición (E. S1 = (10000 | 15000 | 13000))
```

11).- Recuperaciones usando 'Funciones Prefabricadas' y el operador 'Todo'.- Dentro del lenguaje se tienen 5 Funciones Preconstruidas las cuales son: CNT. (Cuenta), SUM. (Suma), PRM. (Promedio), MAX. (Máximo), y MIN. (Mínimo). La forma en que se emplean éstas funciones en las solicitudes, se muestra en los siguientes ejemplos: '"Cuenta" el número total de "Empleados"':

EMP	NOMBRE	
	I. CNT. Todo. E. GARCIA	

Figura 12

Notación Lineal:

Emp (Nombre: I. CNT. Todo. E. García)

En éste tipo de solicitudes la expresión 'Todo. E. García' representa a un conjunto, esto es, que todos los nombres que aparezcan en la relación EMP se consideran como parte de un conjunto o grupo, así entonces, como resultado de esta solicitud se tendrá el numero de elementos que son parte del grupo formado.

12).- Recuperación Restringida usando Funciones Prefabricadas.- 'Imprime la "suma de los salarios" de todos los empleados que trabajan en el departamento de "JUGUETERIA"':

EMP	SAL	DEPT
	I. SUM. Todo. E. S1	C. JUGUETERIA

Figura 13

Notación Lineal:

Emp (Sal: I. SUM. Todo. E. S1, Dept: C. Jugueteria)

Como resultado de ésta solicitud se tendrá la suma de los salarios que están en el grupo que se formó debido a la expresión 'Todo. E. S1', y donde los salarios pertenecen a empleados que laboran en el departamento de juguetería.

13).- Recuperación con Agrupamientos.- 'Para "cada Departamento" imprime el "NOMBRE" de este, y la "Suma de los Salarios" de los empleados que trabajan en él':

EMP	SAL	DEPT
	I. SUM. Todo. E. S1	I. A. E. GOMA

Figura 14

Notación Lineal:

Emp (Sal: I. SUM. Todo. E. S1, Dept: I. A. E. Goma)

Para ésta solicitud, los empleados de la relación EMP se van a agrupar por el Departamento, esto es, se van a generar tantos grupos como departamentos diferentes se tengan, donde los empleados de un determinado grupo pertenecerán al mismo (departamento). Para cada grupo generado se sumarán los salarios de los empleados del mismo.

14).- Recuperación Restringida usando Funciones Prefabricadas y una Caja de Condición.- 'Recupera los "Departamentos" que "cuentan con más de 3 empleados"':

EMP	NOMBRE	DEPT
	Todo. E. GARCIA	I. A. E. GATO

CAJA CONDICION
CNT. Todo. E. GARCIA > 3

Figura 15

Notación Lineal:

Emp (Nombre: Todo. E. Garcia, Dept: I. A. E. Gato)
Caja Condicion (CNT. Todo. E. Garcia > 3)

En ésta solicitud solo se recuperarán los departamentos que cuenten con más de tres empleados como puede ser "GARCIA".

15).- Recuperación que involucra 'conjuntos encadenados que usan el operador 'Todo'.- 'Consigue los nombres de los "Departamentos" donde cada uno de estos vende al menos todos los articulos que vienen en color "VERDE"':

VENTAS	DEPT	ARTI
	I. A. E. PERRO	[Todo. E. PILA, *]

TIPO	ARTI	COLOR
	Todo. E. PILA	C. VERDE

Figura 16

Notación Lineal:

Ventas (Dept: I. A. E. Perro, Arti: [Todo. E. Pila, *])
Tipo (Arti: Todo. E. Pila, Color: C. Verde)

La expresión 'Todo. E. Pila' en la relación TIPO representa a un conjunto formado por los articulos de éste que vienen en color "VERDE". De ésta forma se estará buscando por aquellos grupos, generados al agrupar a la relación VENTAS por departamento, que contengan al menos todos los articulos en color verde. Así entonces el asterisco "*" indica que en los grupos puede haber articulos que no esten en el conjunto 'Todo. E. Pila' (articulos en color verde).

Otra solictud de este tipo viene a ser: 'Consigue los nombres de los "Departamentos" donde cada uno de esto vende todos los articulos que vienen en color "VERDE" y ninguno más:

VENTAS	DEPT	ARTI
	I. A. E. ROPA	Todo. E. PILA

TIPO	ARTI	COLOR
	Todo. E. PILA	C. VERDE

Figura 17

Notación Lineal:

Ventas (Dept: I. A. E. Ropa, Arti: Todo. E. Pila)
Tipo (Arti: Todo. E. Pila, Color: C. Verde)

En este caso se espera que cada uno de los grupos formado de la relación VENTAS al agruparla por departamento, contenga exactamente los mismos artículos que los que se encuentran en el conjunto formado, de la relación TIPO, por la expresión 'Todo. E. Pila'.

Como se mencionó, en éste capítulo se explica y define la sintáxis de Query by Example que se maneja en ésta implantación. En el Apéndice III se da la sintáxis formal de la definición original de QBE.

CAPITULO III

ETAPA GRAFICA

Esta etapa tiene la finalidad de brindarle al usuario la facilidad de generar una consulta a la Base de Datos, usando la nomenclatura correspondiente de "Query by Example" (QBE), así como la de pasar dicha solicitud, que estará en un formato bidimensional, a una forma lineal correspondiente (formato unidimensional), que será lo que entre a la etapa de Generación del Arbol de Análisis Sintáctico.

Dado que el objetivo de éste trabajo es el de la implantación de QBE de la parte de consulta y manipulación de Base de Datos, ésta debe de generarse mediante algún metodo que lo permita. En particular para este trabajo, la base de datos debe de crearse por algún programa en lenguaje Clipper o dBASE III.

El 95% de las rutinas de esta etapa están implementadas en el lenguaje de programación "C", y el 5% restante lo están en "Clipper". Las rutinas en Clipper son las que extraen de las relaciones a usar por parte del usuario para su solicitud, los campos de éstas, para así poder presentárselos en los esqueletos correspodientes.

La etapa gráfica es la primera del Sistema que se ejecuta cuando un usuario desea hacer una consulta a la Base de Datos. Cuando se invoca al Sistema se presenta un Menú mediante el cual se podrá seleccionar los tipos de Esqueletos (de Relación, Caja de Condición o Tabla de Resultados) que se emplearán para definir una consulta. El menú que se presentará, es como se muestra en la Figura III.1:

Mediante éste menú el usuario puede definir hasta cuatro esqueletos de tabla que utilizará para definir la consulta que desea hacer de la base. Esta restricción es debida a las dimensiones del monitor en el que se usa el Sistema.

En el Sistema se tiene contemplados tres tipos de esqueletos de tabla que se pueden utilizar para la definición de consultas. Estos tipos son:

1).- Relacion: Esta tabla estará asociada a una relación de la Base de Datos, y aparecerán en ella, como encabezados de la misma, el nombre y los campos de dicha relación.

2).- Caja de Condición (CC): Esta constará de una sola columna, y tendrá como encabezado la leyenda "CAJA DE CONDICION".

3).- Tabla de Resultados (TR): Este esqueleto tiene el mismo formato que el de una relación. La única diferencia es que los encabezados de las columnas aparecen en blanco, y donde va el nombre de la tabla aparece la leyenda "TR".

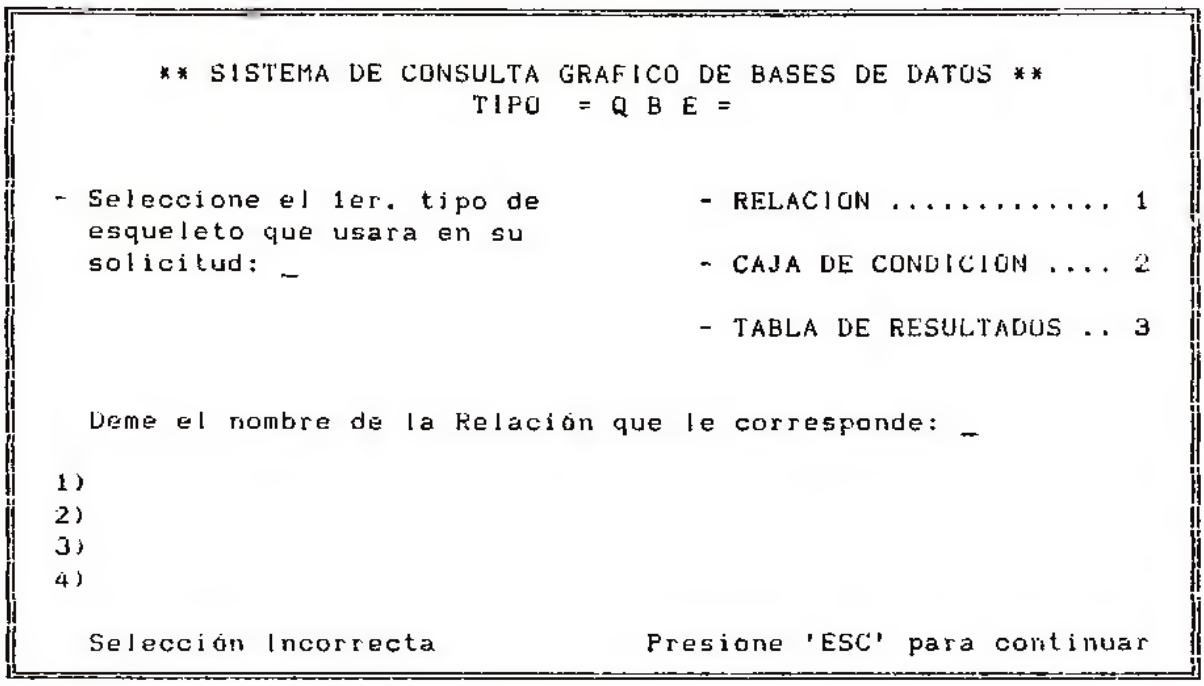


Figura III.1

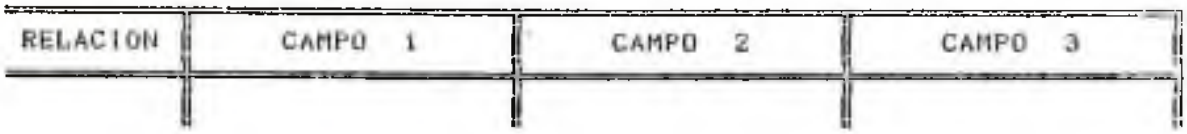
Si el usuario da la opción "1", al indicar el tipo de esqueleto que desea seleccionar, con ésto estará indicando que desea contar con un esqueleto de relación para formular su solicitud. Una vez dada esta opción se le pregunta por el nombre de la relacion que asociará a tal tabla. Si dá la opción "2", significará que necesita tener una Caja de Condición para poder generar su solicitud. Y si da la opción "3" se le presentará el esqueleto de una Tabla de Resultados. En el caso de que dé una opción no contemplada en el menú, le aparecerá el siguiente mensaje de error: "Selección Incorrecta", y se le volverá a pedir que dé otra opción. Si presiona la tecla "ESC" le estará indicando a sistema que ya seleccionó todos los esqueletos que va a emplear, y a continuación se le presentarán en pantalla las tablas correspondientes para que plantee su solicitud. Por cada selección de tipo de esqueleto que realiza, se le presenta en pantalla, el número de tabla seleccionada (1a. tabla, 2a. tabla, etc.) y el tipo de ésta (para una Relacion, el nombre de relación asociado; para una Caja de Condicion, la leyenda "CC"; y para una Tabla de Resultados, la leyenda "TR").

Una vez que el usuario definió ya todo los esqueletos que va a emplear para definir su solicitud, y presionó la tecla "ESC", se transfiere esta información dada, a las correspondientes rutinas en

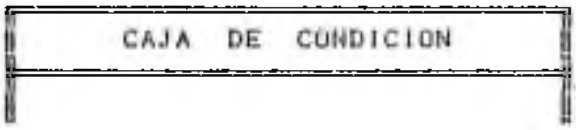
"Clipper", que determinarán los campos que corresponde a cada relación a utilizar, y que deberán aparecer en su respectivo esqueleto de tabla.

Los diferentes tipos de esqueletos con que se cuenta, tienen la siguiente forma:

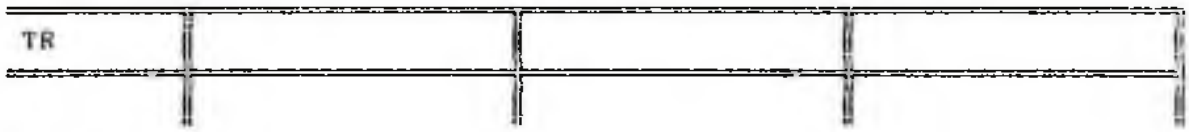
1).- Relación:



2).- Caja de Condición:



3).- Tabla de Resultados:



La forma en que el usuario realiza el llenado de las tablas que se le presentan es:

1) Relación.- Para ésta tabla el cursor se ubicará primero en la línea que se encuentra debajo de los encabezados, en la columna en la que va el nombre de la relación asociado a la tabla. En éste lugar el usuario podrá poner el simbolo "-" si fuera necesario. Presionando la tecla "Enter" salta a la siguiente columna ubicada a la derecha, y de esta forma puede llegar a la(s) columna(s) en la(s) que desee poner alguna expresión de su solicitud. Para indicar al sistema que ha terminado de poner las expresiones que le interesaban en tal tabla, y asi salir de ésta y pasar por ejemplo al siguiente esqueleto a llenar, puede proceder de dos formas:

1a. forma.- Presionando la tecla "ESC" sin importar la columna de la tabla en que se encuentre.

2a. forma.- Presionando "Enter" tantas veces como sea necesario para 'saltar' de la ultima columna (la que esta en la extrema derecha de la tabla) al exterior.

2) Caja de Condición.- Al llegar a esta tabla el cursor se ubica en la linea que se encuentra debajo del encabezado de la misma. En este lugar el usuario puede poner la expresión que le interese. Presionando la tecla "Enter" se sale de ésta tabla.

3) Tabla de Resultados.- La entrada a ésta tabla se hace por la primer columna de los campos en la linea de encabezados. En ésta linea el usuario pondrá los encabezados que desee, de la columnas que ocupará para su solicitud. Para pasar a la linea debajo de la de encabezados, lo puede hacer de dos formas:

1a. forma.- Presionando la tecla "ESC" sin importar la columna en la que se encuentre.

2a. forma.- Presionando "Enter" tantas veces como sea necesario hasta 'saltar fuera' de la ultima columna (la que esta en el extremo derecho de la tabla) de encabezados.

Aquí el cursor aparecerá en la primer columan de los campos (la que está más a la izquierda) en la que se definió un encabezado. Estando en este lugar el usuario podrá definir la expresión de su consulta que le corresponde a tal columna. Presionando la tecla "Enter" pasará a la siguiente columna a la derecha, donde se hará lo correspondiente. Para salir de la tabla se deberá 'saltar a la derecha', presionando la tecla "Enter", de la última columna en la que se puso un encabezado.

Despues de abandonar la última tabla seleccionada, se traduce la solicitud hecha por el usuario a otra equivalente en notación lineal (unidimensional). Esta solicitud ya tiene el formato con que entra a la etapa de "Generación del Arbol Sintáctico". Un ejemplo de esta traducción de notación se muestra a continuación:

Notación Bidimensional de la solicitud: 'Dame los artículos que son de color "VERDE", y son vendidos en departamento de "JUGUETERIA"'.

TIPO	ARTI	COLOR	TAMAÑO
	1. E. RUEDA	C. VERDE	

VENTAS	DEPT	ARTI	
	C. JUGUETERIA	E. RUEDA	

La Notación Lineal equivalente de tal solicitud es:

- Tipo (Arti: I. E. Rueda, Color: C. Verde)

- Ventas (Dept: C. Jugueteria, Arti: E. Rueda)

Por no ser esta fase la más importante de la implementación, se pusieron ciertas restricciones en lo referente a las dimensiones de los esqueletos de las tablas. Estas restricciones son:

i) Solo se pueden emplear hasta cuatro esqueletos para la elaboración de una solicutd.

ii) El número máximo de campos que puede tener una Relación de la Base de Datos es de 3.

Estas restricciones fueron causadas por las dimensiones (número de columnas y lineas) del monitor de la PC para la que se desarrolló el Sistema.

CAPITULO IV

ETAPA DE GENERACION DEL ARBOL SINTACTICO

A continuación se describe el Algoritmo empleado en éste trabajo para pasar de una solicitud en QBE en notación lineal, a un Arbol Binario de operaciones del Algebra Relacional que resuelve una pregunta equivalente.

La elaboración de éste algoritmo tiene como base lo descrito en la referencia número 3.

En la definición del algoritmo se utilizan las operaciones del Algebra Relacional, más algunas que no son propias de ésta. Estas operaciones son implementaciones que se tuvieron que hacer para así poder facilitar la elabaración del algoritmo. La característica básica de estas operaciones extras es que trabajan sobre conjuntos de tuplas y no sobre tuplas aisladas como lo hacen las operaciones 'tradicionales' del algebra relacional. Para el algoritmo las operaciones extra que se implementaron son del tipo: Agrupamiento, Selección de Grupos (Conjuntos); 'Juntas' (Joins) de Grupos; etc.

IV.1 - Descripción del Algoritmo

A ésta etapa la solicitud en QBE hecha por el usuario llega en una notación lineal equivalente, de tal forma que la solicitud estará compuesta por una o más 'Filas ó Tablas'. Cada una de éstas corresponde a una Relación de la Base de Datos a consultar.

Las Operaciones, en total, que se utilizan en el Subsistema se muestran en la Tabla IV.1:

En una forma muy general el subsistema resuelve las operaciones de algebra relacional implicitas en la solicitud en el siguiente orden:

- Selecciones (S)
- Juntas (J)
- Agrupamientos (A)
- Juntas de Grupos (Jg)
- Selecciones de Grupos (Sg)
- Proyecciones (P)

Operaciones consideradas en el Algoritmo.

S $_{condicion\ 1}$ (Relación)	- Selección
(Relación) J $_{condicion\ 2}$ (Relación)	- Join (Junta)
Pr $_{condicion\ 3}$ (Relación)	- Proyección
D $_{Archivo1,\ Archivo2}$	- Diferencia
U $_{Union}$	- Unión
Sg $_{condicion\ 11}$ (Relación)	- Selección de Grupo
(Relación) Jg $_{condicion\ 21}$ (Relación)	- Junta de Grupo
(Relación) SJ $_{condicion\ 2}$ (Relación)	- Semi Junta
Pu $_{condicion\ 4}$ (Relación)	- Proyección Unica
A $_{campo,\ Nada}$	- Agrupamiento
Fp $_{campo\ Funcion\ Pre-fabricada}$ (Relación)	- Función Prefabricada
(Relación1) Re $_{condicion}$ (Relación2)	- Relación Equivalencia

Donde:

[...]	- Indica que los parametros son opcionales. Irán dependiendo de la forma de la solicitud.
< ..., ... >	- Muestra diferentes alternativas que pueden ir.
Condición 1	- Condición [< AND, OR > Condición 1]
Condición	- Campo < =, #, >, <, >=, <= > Constante
Condición 2	- Campo < =, #, >, <, >=, <= > Campo
Condición 3	- Campo-<b, Fun., A>
Condición 4	- Campo [Condición 4]
Condición 11	- Función-Prefabricada Condición
Condición 21	- Conjunto < igual, subconjunto > Conjunto
Conjunto	- Es un conjunto de relaciones (tuplas) que fueron agrupadas por tener un mismo valor en algún determinado campo de la relación.
Re $_{condicion2}$	- Relación1 - Relación1 SJ $_{condicion\ 2}$ Relación2.
Relación1	- Relación ligada a Relación2.
Relación2	- Relación marcada como 'No unible'.
Función Pre-fabricada	- <SUM., CNT., PRM., MAX., MIN>
AND, OR	- Operadores Lógicos.
Constante	- Elemento constante definido en la solicitud
Campo	- Campo de la relación sobre el que se va a operar.

T A B L A IV.1

=,#,>,<,>=,<=	-	Operadores Relacionales.
igual (=), subconjuto (<)	-	Operadores Relacionales para conjuntos.
Campo-	-	Contenido de campo.
Campo-Fun.	-	Resultado de una función pre-fabricada.
Campo-A	-	Contenido del campo de agrupamiento.

T A B L A IV.1
Continuación

IV.2 - Pasos del Algoritmo por análisis sintáctico

Los ejemplos de Solicitudes en QBE que se muestran a lo largo de la descripción del algoritmo están en una notación lineal . Es por esto, que se hablará de 'Tablas' y 'Lineas' indistintamente, y de 'Columnas de Tablas, Entradas y Campos' por igual.

Inicialmente para el Sistema, cada Tabla de la solicitud, constituye una relación completamente independiente de las otras que hubiese en la misma.

Cada Operación del Algebra Relacional que se aplica a alguna(s) relación(es) de la Base de Datos, nos genera una nueva relación que contendrá el resultado de la operación.

Toda solicitud consta de una o más lineas, en donde cada una de estas corresponde a una Relación dentro de la Base de Datos. Dentro de éstas lineas van indicadas las operaciones que se desean realizar sobre éstas relaciones. Asi entonces, el subsistema al tener como entrada una solicitud en QBE realizará las siguiente tareas, que a continuación se detallan, para generar su correspondiente árbol binario de análisis sintáctico. El orden con el que se explican los pasos del algoritmo, corresponde al orden con el que el subsistema 'Analiza' a las Tablas que conforman a la Solicitud que se le pretende generar su árbol.

El árbol se generará de 'Abajo hacia Arriba', donde para cada tabla asociada con una relación (esto es que no sea una Caja de Condición o una Tablas de Resultados), que se emplee en la solicitud se tendrá una Hoja, a partir de la cual se formará una Rama. Conforme se vayan encontrando operaciones del algebra relacional implicitas en la solicitud, se irán creando Nodos del árbol, que vendrán a formar parte de la Rama que correponde a la tabla en la que se detecto tal operación.

El algoritmo está formado de 11 pasos, los cuales son:

1.- Asociación Linea-Relación.
2.- Marcado de lineas 'No Unibles'.
3.- Generación de Selecciones.
4.- Generación de Juntas.
5.- Formación de Grupos.
6.- Generación de Juntas de Grupo.
7.- Generación de Proyecciones Unicas.
8.- Aplicación de Funciones Pre-Fabricadas.
9.- Generación de Selecciones de Grupo.
10.- Generación de Relaciones de Equivalencia.
11.- Generación de Proyecciones.

Los pasos 1 y 2 no implican ninguna operación del algebra relacional. Los pasos 3, 4, 7 y 10 analizan a las operaciones del algebra relacional implícitas en la solicitud que no implican grupos. Los pasos 5, 6, 8 y 9 analizan a las operaciones de grupos implícitas en la solicitud. El paso 11 se aplica por igual tanto a grupos como a relaciones formadas de la base de datos.

A continuación se explican los 11 pasos del algoritmo. Dentro de cada uno:

- Se explicará la tarea que se realiza en éste.
- En el caso que corresponda, se definira y explicará el formato de la operacion del algebra relacional generada en el mismo.
- Se dará(n) un(os) ejemplo(s) de solicitudes que contengan lo visto en dicho paso.

Paso 1.- Asociación Linea-Relación. Una solicitud en QBE está formada por una o más lineas (Tablas), donde cada linea hace referencia a una determinada relación de la base, y a algunos de sus campos.

Ventas (Dept: I. E. Ropa, Arti: E. Rueda)
Surtido (Arti: E. Rueda, Proveedor: C. Parker)

Las Relaciones que se emplearán en esta solicitud son:

- Ventas
- Surtido

Paso 2.- Marcado de lineas 'No Unibles'. Las Tablas que tengan a izquierda del nombre de la relación asociada a la tabla, el símbolo "~" (para éste caso significa 'No existe') se marcarán como No Unibles. Esto significa que no se pueden 'ligar' con alguna otra tabla por medio de la operación de 'Junta', y solo lo podrá hacer por medio de la operación denominada 'Relación de Equivalencia'. Un ejemplo de una solicitud en la que se realiza una operación de Relación de Equivalencia es:

Ventas (Dept: I., Arti: E. Tinta)
~ Surtido (Arti: E. Tinta, Proveedor: C. Parker)

La Tabla en donde está el nombre de la relacion 'Surtido' se marca como 'No unible'.

Nota.- En los ejemplos darán, las Operaciones del Algebra Relacional estarán incompletas, ya que unicamente se presenta lo que se pretende enfatizar, que es el paso del algoritmo explicado.

Paso 3.- Generación de Selecciones. Para cada una de las Tablas dentro de la Solicitud se localizan las Selecciones que existan en ellas. Dentro de una tabla donde se tenga una entrada con un 'Elemento-Constante' (Constante) - donde un elemento de estos tiene la forma 'C. Elemento', se tiene una selección. El formato de esta operación es:

$S_{campo\ operador\ constante}$ (Relación)

Donde:

Campo.- Es el campo de la relación en donde se encuentra la entrada con la constante.
Operador.- Será el operador relacional "=" sino se especifica otra cosa en la entrada, y de lo contrario será el que se aparezca en la misma.
Constante.- Elemento constante definido en el campo.

Un ejemplo de este caso es:

- Tipo (Arti: I. E. Rueda, Color: C. Verde) ==>
  $S_{color\ =\ verde}$ (Tipo)

Si hay más de una entrada con una constante, dentro de una misma tabla, las condiciones de Seleccion se formará como en el ejemplo anterior, y se agruparán por el operandor 'AND', por ejemplo:

- Tipo (Arti: I. E. Bala, Color: C. Verde, Tamaño: C. Mediano) ==>
  $S_{color\ =\ verde\ AND\ tamaño\ =\ mediano}$ (Tipo)

La condición de selección tambien podrá obtenerse de una Caja de Condición que exista en la solicitud. Para este caso la tabla, donde está el campo sobre el cual se va a seleccionar, estará ligada a la caja a través de un Elemento Ejemplo Simple, donde un elemento de estos tienen la forma 'E. Elemento' y no esta precedido por el Operador 'Todo' (esto es, se tiene una entrada en alguna tabla de la solicitud, que tiene un elemento ejemplo simple que aparece tambien dentro de la caja de condición). En este caso se tiene que existe más de una condición de selección para un cierto campo de una relación, lo que no es facil de expresar en una columna del esqueleto de tabla sobre el cual el usuario realiza su solicitud. Las diferentes condiciones de selección se asociarán por los Operadores lógicos 'AND' (&) u 'OR' (;), según se indique en la caja. En estos casos se pone una condición para cada una de las constantes de la caja, teniendo como operador, al que venga con la misma, en caso de que se indique, y de lo contrario se considera al operador '='. Estas constantes se aplican al campo en donde aparece el elemento ejemplo simple. Esto se muestra en el siguiente ejemplo:

- Emp (Nombre: I. , Sal: E. S1)
  Caja Condicion (E. S1 = (> 10000 & < 15000 & ~ 13000))

  ==>

  $S_{Sal > 10000 \text{ AND } Sal < 15000 \text{ AND } Sal \sim 13000}$ (Emp)

- Emp (Nombre: I. , Sal: E. S1)
  Caja Condicion (E. S1 = (1000 ! 13000 ! 16000))

  ==>

  $S_{Sal = 10000 \text{ OR } Sal = 13000 \text{ OR } Sal = 16000}$ (Emp)

Paso 4.- Generación de Juntas. Cuando exista más de una Tabla en una solicitud se podrá tener el caso de que dos de estas tablas estén 'ligadas' por la aparición de un mismo Elemento Ejemplo simple. Esto es, en una de las tablas existe una entrada con un elemento ejemplo que aparece en alguna entrada de la otra tabla. Cuando esto suceda, se genera una operación de Junta (J) del Algebra Relacional. En ésta Junta los campos sobre los que se realizará la operación, serán aquellos en los que aparece el Elemento Ejemplo simple común. El Formato de esta operación es:

(Relación 1) $J_{condicion}$ (Relación 2)

Donde:

Relación 1 y Relación 2.- Son relaciones de la base cuyas tablas asociadas, presentan entradas ligadas.
Condición.- Es la condición mediante la cual se va a realizar la junta. El operador de esta relación será el que se especifique en cualquiera de las entradas, y en caso de que no suceda así se considerará al operador '='.

Lo anterior se muestra en el siguiente ejemplo:

- Ventas (Dept: C. Jugueteria, Arti: E. Tuerca)
  Tipo (Arti: I. E. Tuerca, Color: C. Verde)

  ==>

  $(S_{Dept = Jugueteria}$ (Sales)) $J_{Arti = Arti}$ $(S_{Color = Verde}$ (Type))

Paso 5.- Formación de Grupos. Se forman los Grupos de aquellas relaciones en donde se indique. En aquellas tablas en las que en algunas de sus entradas exista el operador "Todo.", significará que esa relación, o la generada a partir de esa, deberá de agruparse (dividirse en grupos). Los casos bajo los que se puede indicar y realizar el agrupamiento son tres:

Caso i).- Exista otra entrada con el operador de agrupamiento "A.", a la izquierda de un elemento ejemplo simple (A. E. Jugueteria), lo que indica que tal relación debera de agruparse por el campo en el que ocurrio este operador.

Caso ii).- No se tenga alguna entrada en donde aparezca el operador de agrupamiento, lo que indica que esta relación debe de agruparse por 'Nada', esto es, que toda la relación debe de tratarse como un grupo.

El formato de esta Operación es:

A_{campo} (Relación)

Donde:

Campo.- Es el campo de la entrada en el que aparece el operador "A.". Si el agrupamiento se hace por la ocurrencia de la situación indicada en el caso 'ii', entonces irá la palabra 'Nada'.
Relación.- La relación a agrupar.

Lo anterior se muestra en el siguiente ejemplo:

- Tipo (Arti: Todo. E. Tinta) ==> A_{Nada} (Tipo)

- Tipo (Arti: Todo. E. Tinta, Color: C. Verde)

  ==>

  A_{Nada} ($S_{Color = Verde}$ (Tipo))

- Emp (Nombre: Todo. E. Juan, Dept: I. A. E. Regalos)

  ==>

  A_{Dept} (Emp)

Paso 6.- Generación de Juntas de Grupo. Si existen en la solicitud dos tablas con entradas que contienen al Operador "Todo." y además un mismo elemento ejemplo, se realiza una operación de Junta de Grupos (Jg). Esta operación tiene la siguiente formato:

Relación1 $Jg_{condicion}$ Relación2

Donde:

Condición.- Grupo1 < 'C', '=' > Grupo2
C.- Operador de Subconjuntos. Grupo1 está contenido en el Grupo2
=.- Operador de igualdad de conjuntos. Grupo1 está contenido en el Grupo2, y viceversa.

Cuando se tengan dos tablas con las características antes mencionadas, se pueden tener los siguientes dos casos:

Caso i).- Un 'Todo' entre parentesis cuadrados y el otro no, ésto es que se tenga: "[Todo. E. Jose, *]" y "Todo. E. Jose".

Caso ii).- No se tenga alguna ocurrencia de parentesis cuadrados en cualquiera de las tablas, es decir que se tenga algo como: "Todo. E. Jose" y "Todo. E. Jose".

Será condición indispensable que cualquiera de las dos relaciones este agrupada por algún determinado campo de las misma.

Como se mencionó, cuando en una tabla aparece un 'Todo', significa que en esa relación se debe efectuar un agrupamiento por algún campo o por 'Nada'. Asi para el caso i) el grupo generado de la relación con la entrada en parentesis contendrá a todos los elementos de un grupo formado de la otra relación, y posiblemente algunos no contenidos en esta última. Para el caso ii) se tiene que ambos grupos deben de contener exactamente los mismos elementos. Todo lo anterior se muestra en los ejemplos siguientes:

- Ventas (Dept: I. A. E. Tina, Arti: Todo. E. Tinta)
  Tipo (Arti: [Todo. E. Tinta, *], Color: C. Verde)

  ==>

  (A $_{Dept}$ (Ventas)) Jg $_{Arti \subset Arti}$
  (A $_{Nada}$ (S $_{color = verde}$ (Tipo)))

- Ventas (Dept: I. A. E. Jugueteria, Arti: Todo. E. Tinta)
  Tipo (Arti: Todo. E. Tinta, Color: C. Verde)

  ==>

  (A $_{Dept}$ (Ventas)) Jg $_{Arti = Arti}$
  (A $_{Nada}$ (S $_{color = verde}$ (Tipo)))

De los ejemplos anteriores es importante hacer notar que:

- El grupo obtenido de la relación agrupada por 'Nada', permanecerá fijo, esto es, los grupos generados de la otra relación se compararán siempre contra éste.

- Los grupos obtenidos de la relación que se agrupó por algún determinado campo de la misma, se irán comparando uno a uno con el grupo (único) proveniente de la otra relación.

- Los grupos que cumplieron con la condición se seguirán usando, esto es, se les aplicarán las operaciones que sigan, mientras que los que no cumplieron se desechan.

Paso 7.- Generación de Proyecciones Unicas. Se realiza una Proyección Unica, sobre la relación correspondiente de aquellos campos que en la solicitud tengan: El Operador "I." y/o tengan un operador de Función Pre-fabricada. Si la relación fué agrupada por algún campo de la misma, o por 'Nada' no se la aplica este paso del algoritmo. Esta proyección intermedia tiene como finalidad, el eliminar posibles 'Registros repetidos' que se pudiesen tener como resultado de la solicitud. Las funciones que consideran en la implementación son:

CNT. -> Cuenta
SUM. -> Suma
PRM. -> Promedio
MAX. -> Máximo
MIN. -> Mínimo

La Operación de 'Proyección Unica' tiene el siguiente formato:

$Pu_{Campo(s)}$ (Relación)

Donde:

Campo(s) - Son los campos a proyectar de la relación.

Algunos ejemplos de estas solicitudes con estos casos son:

- Tipo (Arti: I., Color: C. Verde)

==>

Pu_{Arti} ($S_{Color = Verde}$ (Tipo))

Según la gramática definida para QBE, solo una tabla de la solicitud puede tener especificado el operador "I.", y este puede aparecer en cualquier entrada de la misma. Cuando se requiere generar resultados de diferentes relaciones de la solicitud, ésto se debe hacer a traves de una Tabla de Resulados. Esta tabla tendrá tantas entradas com proyecciones de campos se desee hacer. Cada entrada de la tabla tendrá la palabra clave "I." y un elemento ejemplo simple, que ligará a ésta, con una entrada de alguna relación definida en la solicitud. Un ejemplo de este caso es:

Ventas (Dept: E. Ropa, Arti: E. Tinta)
Surtido (Arti: E. Tinta, Proveedor: E. IBM)
Tabla Resulta (Cosas: I. E. Ropa, XXX: I. E. IBM)

**>

Pu dept proveedor ((Sales) J arti = arti (Surtido))

Paso 8.- Aplicación de Funciones Pre-Fabricadas. Se analízan las "Funciones Pre-Fabricadas" (CNT, SUM, MAX, etc.), que se tengan. Esta función puede estar asociada a una realación en cualquiera de las siguientes dos formas:

i.- Que aparezca explicitamente en la tabla de la relación. Siempre que esto suceda la función deberá ir precedida de la palabra clave "I.". En éste caso lo que se está indicando es que se desea obtener como salida de la solicitud, el resultado que arroje la función, al aplicársele a cada uno de los grupos en que se haya dividido tal relación; por ejemplo:

a).- Emp (Sal: I. SUM. Todo. E. Si, Dept: I. A. E. Regalos)

ii.- Puede aparecer dentro de una Caja de Condición, y el elemento ejemplo de la caja ligará a esta con alguna entrada de una relación. Cuando esto ocurre, significa que el resultado que se genere de aplicar la función a cada grupo en que se dividió la relación, se utilizará como operando de una condición para seleccionar a tal grupo (Seleccion de Grupos) o desecharlo:

b).- Emp (Name: Todo. E. Pedro, Dept: I. A. E. Deportes)
Caja Condición (CNT. Todo. E. Pedro > 3)

El campo de la relación sobre el que se aplica la función es: el de la entrada con la función, para el caso 'i'; o el que tienen la entrada ligada a la caja de condición, con la función en ella, como en el caso 'ii'. El formato de la operación que generan éstas funciones es:

Fp campo <Funcion Pre-fabricada> (Relación)

Donde:

<Función Pre-fabricada>.- El el operador de la función que se desea aplicar.
Campo (Relación).- Es el campo de relación sobre el que se aplicará la función.

Las operaciones del algebra relacional que se generan para los ejemplos dados son:

a).- FP $_{sal\ SUM}$ (A $_{Dept}$ (Emp))

b).- FP $_{Nombre\ CNT}$ (A $_{Dept}$ (Emp))

Paso 9.- Generación de Selección de Grupos. Para que se efectúe ésta operación es necesario que exista una tabla que se haya agrupado por alguno de sus campos y que tenga alguna entrada ligada a la Caja de Condición. La condición que aparecerá como operador de selección en ésta operación será la que aparezca en la caja, y se aplicará al campo, de la realción, con el que se realiza la liga. El formato de ésta Operación es:

Sg $_{condicion}$ (Relación)

Donde:

Condición.- Es la que aparece en al caja de condición.
Relación.- Es la que tiene al campo cuya entrada se encuentra ligada a la caja.

Un ejemplo de una solicitud que contiene una operación de éste tipo es:

Emp (Nombre: Todo. E. Jimenez, Dept: I. A. E. Discos)
Caja Condición (CNT. Todo. E. Jimenez > 3)

==>

Sg $_{CNT.\ Nombre\ >\ 3}$ (Fp $_{Nombre\ CNT}$ (A $_{Dept}$ (Emp)))

Paso 10.- Generación de Relaciones de Equivalencias. Esta operación se tiene cuando alguna tabla de la solicitud fué marcada como no unible. Esta relación deberá estar ligada a otra por medio de un elemento ejemplo simple común. Esta Operación tiene la siguiente sintáxis:

(Relación 1) Re $_{condicion}$ (Relación 2) =
Relación 1 - (Relación 1 SJ $_{condicion}$ Relación 2)

Donde:

SJ.- Operación de Semi Junta. Una operación de estas es una 'Junta' con una Proyección sobre la relación resultante, de los campo de la Relación 1.
Condición.- Es la condición sobre la que se realizará la operación de 'Junta'.
Relación 2.- Relación marcada como no unible.

La condición de la Relación de Equivalencia se formará de la misma forma que la condición para la 'Junta', esto es: los operandos

serán los campos en los que está el elemento ejemplo simple común a las relaciones, y el operador será el que se especifique con cualquiera de los elemento ejemplo. se supondrá '=' si no se especifica alguno).

Un ejemplo de una solicitud con esta operación es:

Ventas (Dept: I., Arti: E. Tinta)
  Surtido (Arti: E. Tinta, Proveedor: C. Parker)

==>

(Ventas) Re $_{Arti = Arti}$ (S $_{Proveedor = Parker}$ (Surtido))

Paso 11.- Generación de Proyecciones. Se consideran las Proyecciones (Pr) que nos darán las salidas esperadas de la solicitud. Todas las entradas que tengan la palabra clave "I." producirán una 'Proyección' del campo en el que se encuentra tal entrada.

El formato de la Proyección es:

Pr $_{Campo(s)}$ Relación

Donde:

Campo(s).- Son los campos de las entradas en los que aparece el operador "I.".
Relación.- Es la relación en la que están los campos a proyectar.

La información que se desea proyectar de la entrada en el que se definió el operador "I.", dependerá de que otros operadores aparezcan en la misma. Dado lo anterior, se tienen los siguientes tres casos:

Caso i).- El elemento ejemplo de la entrada en que se tiene el operador "I." es simple, o se encuentra solo en la entrada. Esto significa que lo que se desea proyectar es el contenido del campo de la relación que corresponde a dicha entrada; por ejemplo:

Ventas (Dept: I. E. Nada) ==> Pr $_{Dept.}$ (Pu $_{Dept}$ (Ventas))

También dentro de este caso estan aquellas solicitude en las que usa una Tabla de Resultados, por ejemplo:

Ventas (Dept: E. Jugueteria, Arti: E. Tinta)
Surtido (Arti: E. Tinta, Proveedor: E. IBM)
Tabla Resulta (Cosas: I. E. Jugueteria, XXX: I. E. IBM)

==>

Pr $_{Dept-Supplier}$ (Pu $_{Dept\ supplier}$ ((Ventas) J $_{Item = Item}$ (Surtido)))

Caso ii).- El operador "I." se encuentra en una entrada que también tiene el operador "A.". Entonces no se proyectará el campo de la relación correspondiente a la entrada, sino que se proyectará tal campo de aquellas relaciones formadas al efectuar el agrupamiento. Una solicitud con este caso se muestra a continuación:

Ventas (Dept: I. A. E. Tuerca, Arti: [Todo. E. Tinta, *])
Tipo (Arti: Todo. E. Tinta, Color: C. Verde)

==>

Pr $_{Dept-A}$ (A $_{Nada}$ (S $_{Color = Verde}$ (Tipo)))
Jg $_{Arti\ c\ Arti}$ (A $_{Dept}$ (Ventas))

Caso iii).- El operador "I." está en una entrada en la que aparezca alguna Función Pre-fabricada. Entonces se obtendrá el resultado de aplicar la función sobre el campo correspondiente. Un ejemplo de una solicitud que comprende este caso se muestra a continuación:

Emp (Sal: I. SUM. Todo. E. Si, Dept: C. Jugueteria)

==>

Pr $_{Sal-SUM}$ (Fp $_{Sal}$) $_{SUM}$ (A $_{Nada}$ (S $_{Dept = Jugueteria}$ (Emp))))

Por la gramática de QBE, tenemos que se puede tener al operador "I." en más de una entrada de una tabla. Estas entradas pueden ser combinaciones de los casos anteriormente descritos (elementos ejemplo simple, campos agrupados, etc.). Así entonces, se puede tener que las proyecciones a realizar sobre una tabla de la solicitud, no se generen sobre la misma relación, por ejemplo:

Emp (Sal: I. SUM. Todo. E. Si, Dept: I. A. E. Abarrotes)

IV.3 - Implantación del Algoritmo al Sistema.

El programa que define el algoritmo al sistema, está hecho en el Lenguaje "C". Este programa nos genera un Arbol Binario de Análisis Sintáctico donde cada nodo del mismo, define una operación del Algebra Relacional, que se ha de aplicar sobre alguna(s) relación(es) de la Base de Datos.

La Estructura de Datos que se usa para representar a cada nodo del árbol, es el "Registro" (o estructura de acuerdo a la notación de C). Se tiene un tipo de registro para las Hojas del árbol, y otro para los Nodos. Para cada Tabla de la solicitud se genera una hoja del árbol, y de aquellas que tengan asociada una relación 'crecerá' una rama del mismo; los nodos de la rama se irán generando conforme se vayan analizando las tablas de la solicitud y se detecten, implícitas a ella, operaciones del Algebra Relacional.

Los campos principales del Registro 'Nodo Hoja' son:

Relación.- Es el nombre de la relación que le corresponde a la tabla asociada a la hoja.
Entradas.- Contiene a las entradas de la tabla.

Los campos principales del Registro 'Nodo no Hoja' son:

Apuntador Padre.- Apunta a su Nodo Padre.
Hijo Derecho e Izquierdo. - Apuntadores a los hijos derecho e izquierdo respectivamente.
Operación.- Indica la operación del algebrea relacional que le corresponde al nodo.
Operandos.- Contiene a los Operandos de la operación.
Relación Salida.- Es el nombre de la relación que se tiene como resultado de aplicar la operación, sobre alguna(s) relación(es).

El subsistema de generación del árbol determinará el contenido de cada uno de los campos de los diferentes tipos de registro mencionados, a excepción del campo 'Relación Salida' cuyo contenido se conocerá hasta que se realice el Recorrido del Arbol (etapa de la que se hablará más adelante).

La relación sobre la que se aplicará la operacion del algebra relacional especificada, en el campo correspondiente, en cada nodo, será(n) la(s) que se tenga en el campo 'Relación Salida' del(os) nodo(s) hijo, a excepción de cuando el hijo sea una hoja, en cuyo caso la operación se aplicará sobre aquella relación que se especifique en el campo 'Relación'.

A continuación se dá la representación gráfica de los árboles que se generarían para algunas solicitudes realizadas a una Base de

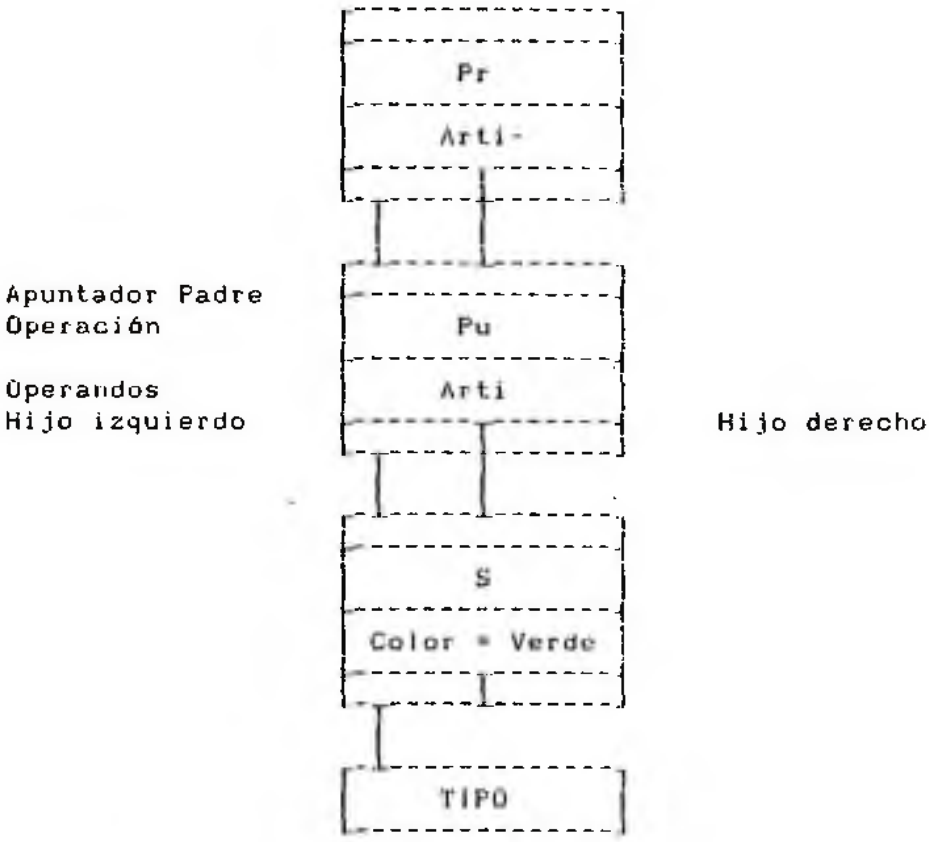
Datos, en notación de QBE. Para ir de acuerdo a lo manejado en el algoritmo, la solicitud se presentará en su notación equivalente lineal:

1.- 'Muestra todos los articulos que vienen en color "Verde"'.

- Solicitud en Forma Lineal:

Tipo (Arti: I. E. Rueda, Color: C. Verde)

- Arbol de Análisis Sintáctico:

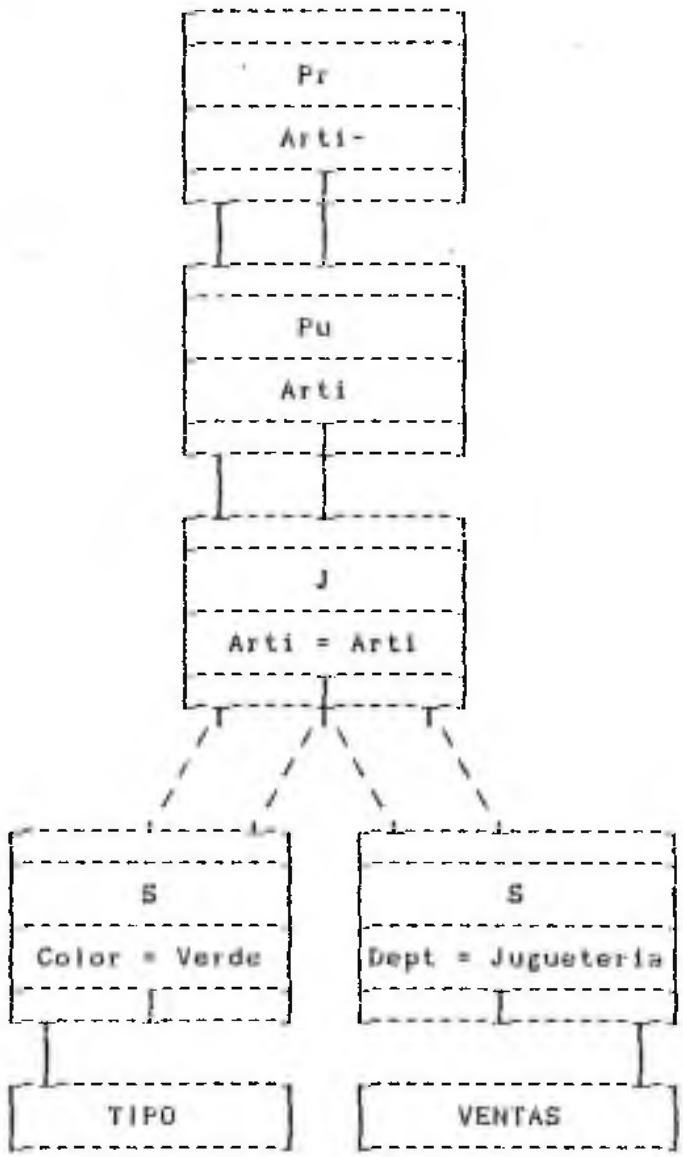


2.- 'Imprime todos los artículos que vengan en color "Verde", y sean vendidos por el departamento de "Jugueteria"'.

- Solicitud en Forma Lineal:

Tipo (Arti: I. E. Nuez, Color: C. Verde)
Ventas (Dept: C. Jugueteria, Arti: E. Nuez)

- Arbol de Análisis Sintáctico:

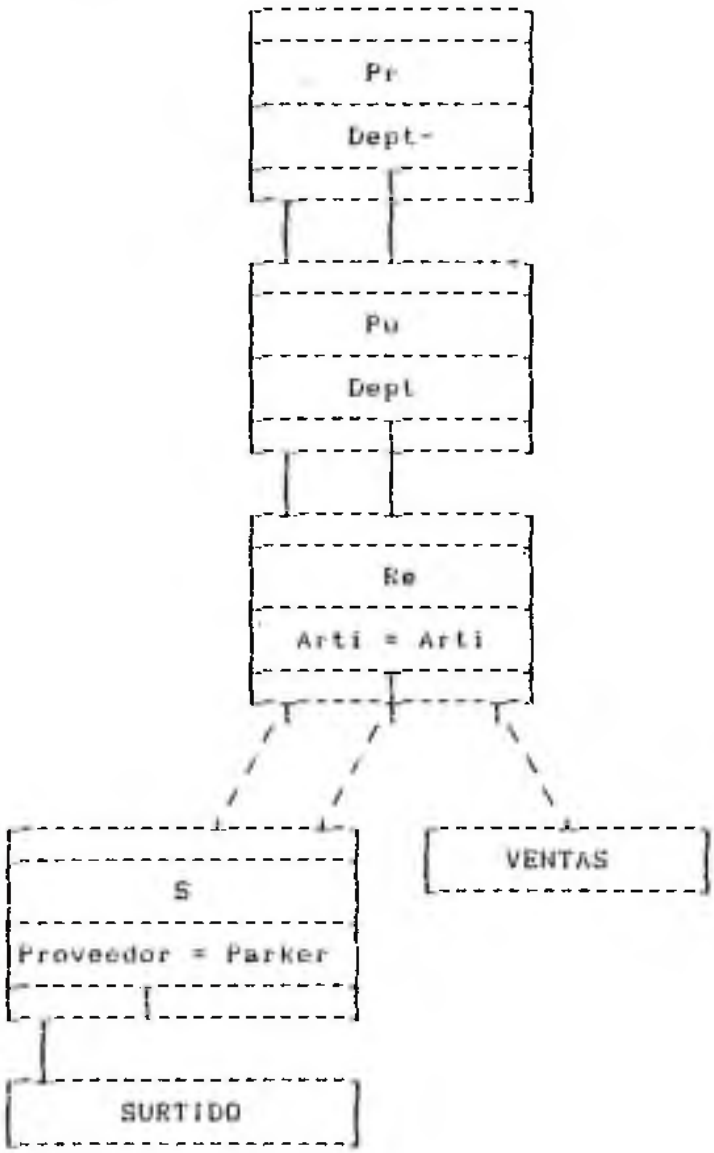


3.- 'Dá los Departamentos que no vendan algún articulo provisto por "Parker"'.

- Solicitud en Forma Lineal:

~ Surtido (Arti: E. Tinta, Proveedro: C. Parker)
Ventas (Dept: I., Arti: E. Tinta)

- Arbol de Análisis Sintáctico:

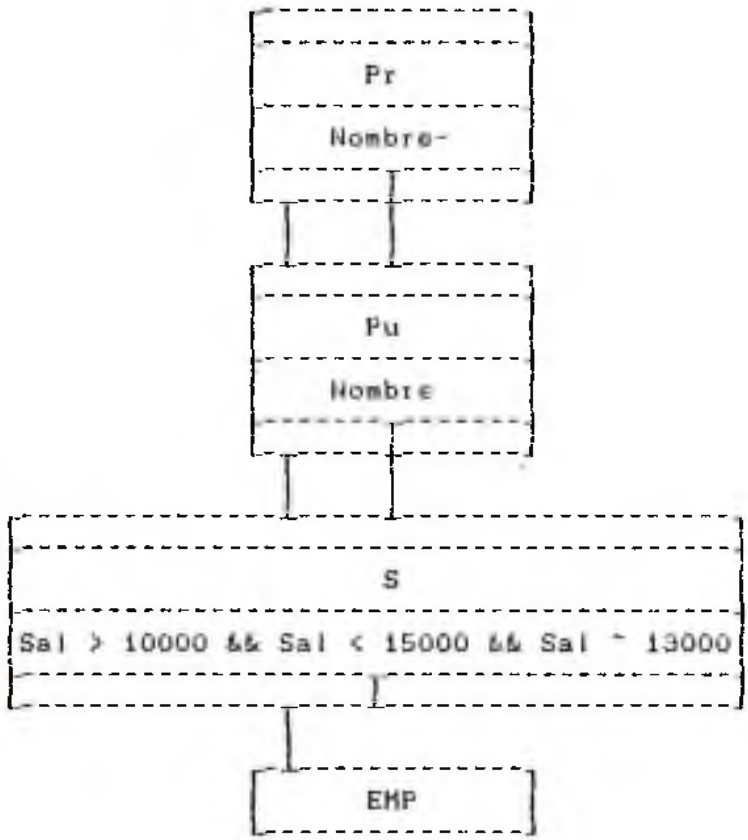


4.- 'Imprime el "Nombre" de los empleados cuyo salario sea: mayor de "$ 10,000", menor de "$ 15,000", y diferente de "$ 13,000"':

- Solicitud en Forma Lineal:

Emp (Nombre: I., Sal: E. S1)
Caja Condición (E. S1 = (> 10000 & < 15000 & ~ 13000))

- El Arbol de Análisis Sintáctico:

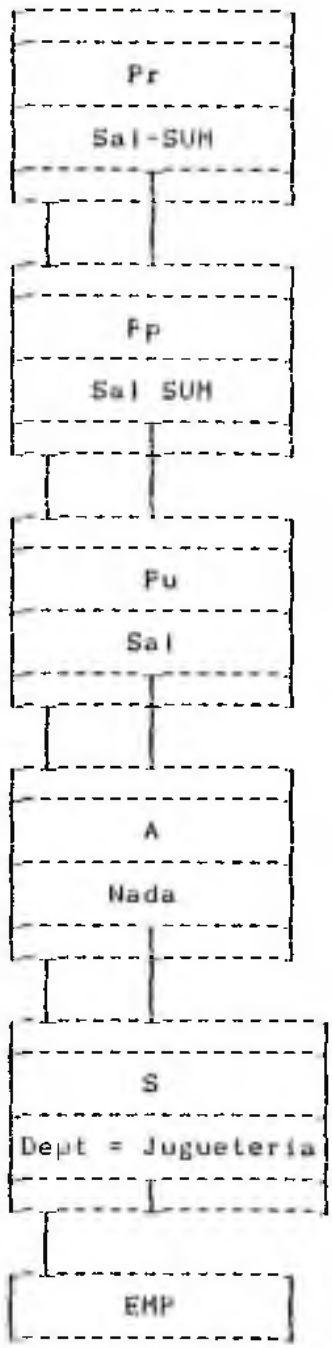


5.- 'Dame la "Suma" de los salarios de los empleados que laboren en el departamento de "Jugueteria"'.

- Solicitud en Forma Lineal:

Emp (Sal: I. SUM. Todo. E. Sl, Dept: C. Jugueteria)

- Arbol de Análisis Sintáctico:

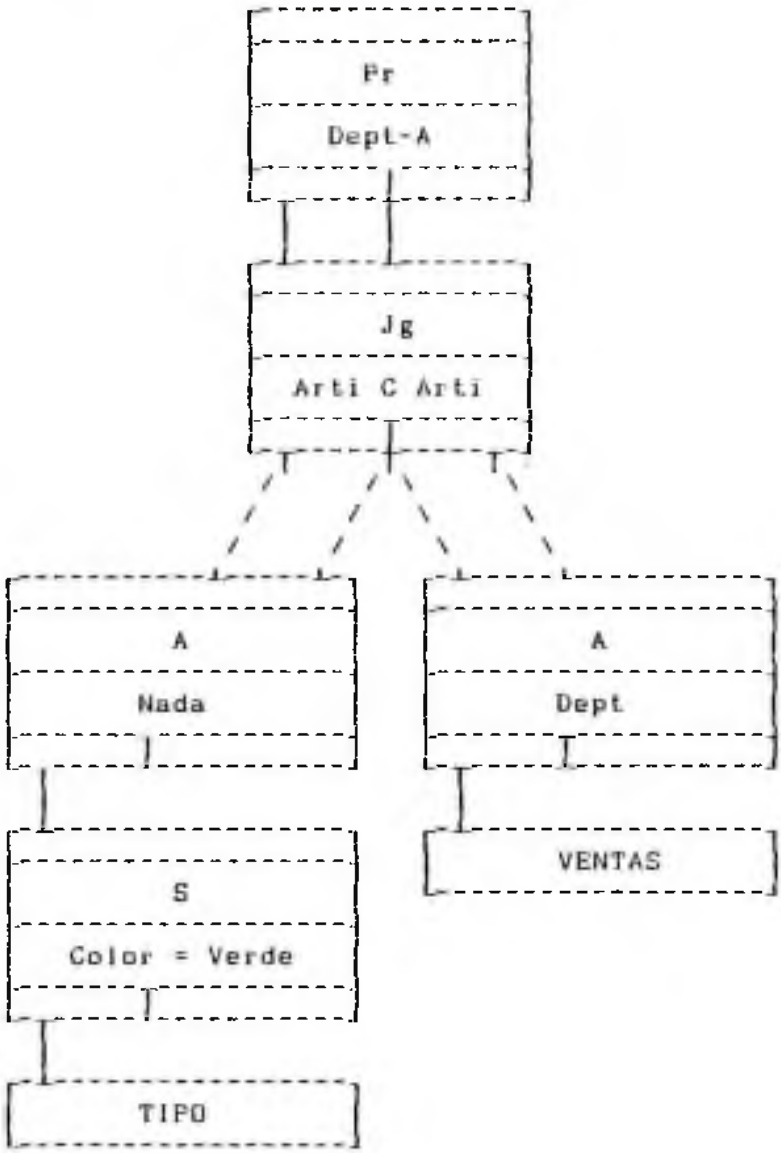


6.- 'Consigue los nombres de los Departamentos que venden al menos todos los artículos que vienen en color "Verde"'.

- Solicitud en Forma Lineal:

Ventas (Dept: I. A. E. Sosa, Arti: [Todo. E. Tinta, *])
Tipo (Arti: Todo. E. Tinta, Color: C. Verde)

- Arbol de Análisis Sintáctico:

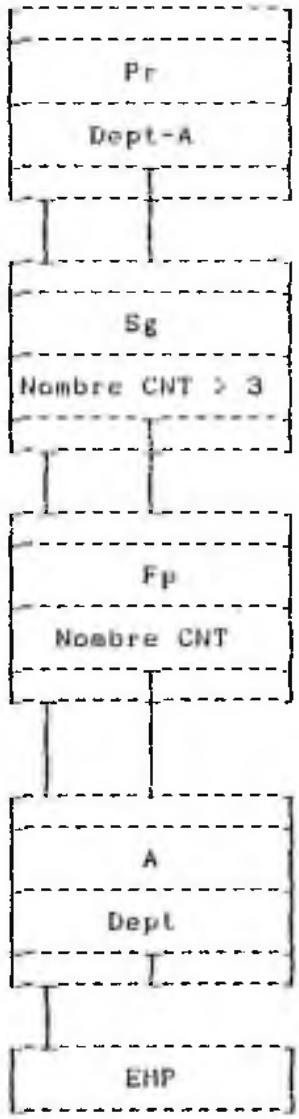


7.- 'Consigue todos los departamentos que tienen más de 3 empleados'.

- Solicitud en Forma Lineal:

Emp (Nombre: Todo. E. Juárez, Dept: I. A. E. Pan)
Caja Condición (CNT. Todo. E. Juárez > 3)

- Arbol de Análisis Sintáctico:



CAPITULO 5

ETAPA DE RECORRIDO DEL ARBOL

En esta etapa se recorre el Arbol Binario de Análisis Sintáctico creado en la etapa anterior del Sistema y se llaman a las correspondientes rutinas de manipulación de la Base de Datos.

El programa de esta etapa está hecho en el Lenguaje "C", y en "Clipper". Las rutinas en C efectúan el recorrido del árbol y en las rutinas en Clipper están construidas las operaciones del algebra relacional que se contemplan en el sistema. Cada vez que un Nodo (no-terminal) del árbol es alcanzado se realiza un llamado a una rutina en Clipper, que efectuará la operación algebraica correspondiente sobre la relación indicada en el(los) nodo(s) hijo(s). El 30% de la programación de esta fase está hecha en C, y el 70% restante en Clipper.

El recorrido del árbol se realiza en 'Pre-orden'. Esto se hace en los siguientes 9 pasos:

Paso 1.- La primer rama (Rama 1) a recorrer será la que corresponda a la primer tabla, que tenga asociada una relación, dada en la solicitud. En general, la segunda rama (Rama 2) del árbol será aquella que corresponda a la segunda tabla dada de la solicitud con una relación asociada, y así sucesivamente.

Paso 2.- De la rama seleccionada, se alcanza el primer nodo, no hoja, de la misma (que viene a ser el padre de la hoja).

Paso 3.- De la estructura que corresponde al nodo alcanzado, se obtiene la Operación del Algebra Relacional así como los Operandos de la misma. La relación sobre la cual se va a operar será:

- La relación que está asociada con la tabla que le corresponde a la rama si la hoja bajo análisis es el primera de la misma:
- Siendo cualquier otro nodo, no hoja, de la rama, la relación será la que se especifique en el campo 'Relación Salida' de su nodo hijo.

Paso 4.- Se analiza el siguiente nodo de la rama, que es el Nodo Padre del recientemente analizado, y se procede como en el paso 3.

Paso 5.- De la rama bajo análisis se prosigue como en el punto 4, tratando de recorrer todos los nodos de la rama, hasta que:

- Se alcance el último nodo de la rama, que vendría a ser el Nodo Raíz de la misma.
- Se llegue a un nodo del que parten dos ramas, es decir, que tiene dos hijos.

Paso 6.- Cuando se llega a algún nodo con dos descendientes generado por una operación binaria, se suspende momentaneamente el análisis de este nodo, y se alcanza al Nodo Hoja de la otra Rama que parte del mismo.

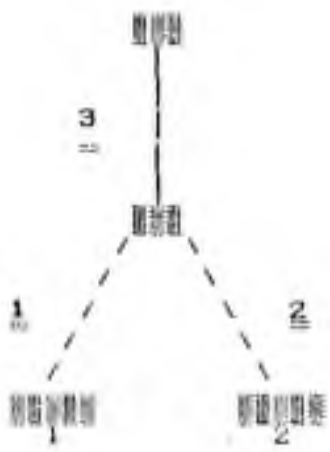
Paso 7.- Una vez alcanzado el nodo hoja de la segunda rama, que puede ser por ejemplo la Rama 2, se prosigue como en los pasos 3 y 4, hasta llegar, nuevamente, al nodo binario.

Paso 8.- Ya que se ha alcanzado al nodo binario por su segunda rama, se obtiene la operación que le corresponde, así como sus operandos. La relaciones sobre la que se va a operar, serán aquellas que se especifique en el campo 'Relación Salida' de su(s) hijo(s) o que esté(n) asociado(s) a su(s) rama(s).

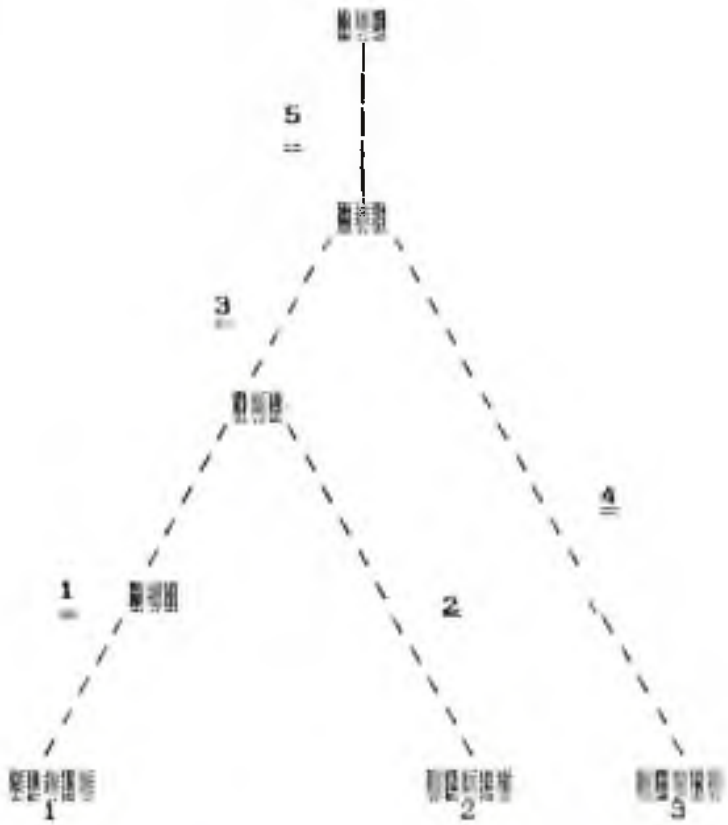
Paso 9.- Se prosigue como se indica en los pasos 2 al 8 hasta que no quede alguna rama por recorrer.

A continuación se muestran algunos ejemplo de la secuencia en que se recorrerían las Ramas de un Arbol Binario, de acuerdo a este Algoritmo:

i).-



ii).-



Donde:

[illegible] - Simbología usada para representar a un Nodo Hoja.

X - Número de Rama del Arbol.

[illegible] - Simbología usada para representar a un Nodo del Arbol.

Y - Secuencial que indica el orden de recorrido de cada Sub-Rama del Arbol.

Cada vez que se alcanza un Nodo intermedio del árbol, los parámetros que se le pasan a las rutinas que realizan el Manejo de la base de datos para ejecutar sobre la(s) relación(es) que se indica(n) la operación del algebra que en dicho nodo se define son:

- Operación a realizar.
- Los Operandos.
- La(s) Relación(es) sobre la que se operará.

Una vez que se realizó la operación correspondiete sobre la(s) relación(es) de la base de datos, el parámetro que se le devuelve a las rutina que reaiza el recorrido es:

- Relación resultante de aplicar la operación.

Las rutinas sobre las que se implementaron las Operaciones del Algebra Relacional, y que consecuentemente realizan la Manipulación de la Base de Datos, requieren que existan, al momento que se invoca al Sistema para realiza una consulta:

- Aquellas Relaciones que potencialmente pueden ser usada para que el usuario realice una consulta, y que conforman la Base de Datos.

- La Relación que define la Estructura de un Archivo de Estructura de Relación. Estas estructuras tienen el formato que se definen en Clipper. Dentro del Sistema, esta relación se denomina: "Structure.dbf".

La implantación de las operaciones del algebra relacional no fué trivial, a pesar de que en el lenguaje Clipper se tienen algunos comandos que se pueden considerar que realizan estas operaciones. Dado que para la mayoria de las operaciones el número de argumentos definidos en el parámetro "operandos" es variable, fué necesario realizar un "Analisis Léxico/Sintáctico" de este. Además con el fín de optimizar espacio en el dispositivo de almacenamiento secundario empleado, fué necesario adoptar medidas especiales para esto.

REFERENCIAS Y BIBLIOGRAFIA

1).- C. J. Date, "An Introduction to Database Systems" Volumen I, 3a. Edición. The Systems Programming Series.

2).- M.M. Zloof, "Query-by-Example: A Data Base Language." IBM Sys. J. 16, No. 4 (1977).

3).- Dennis McLeod, "The Translation and Compatibility of Sequel and Query By Example", IBM Research Laboratory San Jose Ca. Presentado en "the 2nd. International Conference and Software Engineering", San Francisco CA, 13-15 de Octubre de 1976.

4).- Jeffrey D. Ullman, "Principles of Databases Systems", 2a. Edición. Computer software engineering series: Computer Science Press.

5).- M. M. Zloof, "Security and Integrity within the Query-by-Example Data Base Management Language. IBM Thomas J. Watson Research Report.

6).- Samuel P. Harbison and Gay L. Steele Jr., "C a reference manual": Tartan Laboratories. Prentice-Hall, Inc.

7).- Brian W. Kernighan and Dennis M. Ritchie "The C programming Language": Bell Laboratories, Murray Hill New Jersey. Prentice-Hall, Inc.

8).- C. T. Zahn, "C Note: A Guide to the C Programming Language". Yourdan Press.

9).- Manual de Referencia de "Lattice 8086/8088 C Compiler", Version 2.00. Lattice, Inc.

10).- Technical Bolletin de "Lattice MS-DOS C Compiler", Version 3.00. Lattice, Inc.

11).- User's Guide "Microsoft MS-DOS Operating Systems", Version 3.1. Microsoft Coorporation.

12).- User Manual "dBASE III" Version 1.1. Asthon-Tate.

13).- Manual de "Using dBASE III Plus". Asthon-Tate.

14).- "Clipper Compiler User Manual". Nantucket, Inc.

15).- M. M. Zloof, "The syntax of Query-by-Example".

APENDICE I

BASE DE DATOS Y RESULTADOS DE LOS EJEMPLOS DE SOLICITUDES.

Las Relaciones consideradas en esta Base de Datos ejemplo son:

I).-

EMP	NOMBRE	SAL	DEPT
	GARCIA	8000	HOGAR
	PEREZ	6000	JUGUETERIA
	MACHUCA	10000	COSMETICOS
	SANCHEZ	12000	PAPELERIA
	RANGEL	6000	JUGUETERIA
	CISNEROS	16000	COSMETICOS
	JUAREZ	7000	COSMETICOS
	CHAVEZ	8000	HOGAR
	JUNCUA	12000	PAPELERIA
	OLGUIN	9000	JUGUETERIA

II).-

VENTAS	DEPT	ARTI
	PAPELERIA	PLATO
	HOGAR	PLUMA
	PAPELERIA	LAPIZ
	COSMETICOS	LABIAL
	JUGUETERIA	PLUMA
	JUGUETERIA	LAPIZ
	JUGUETERIA	TINTA
	COSMETICOS	PERFUME
	PAPELERIA	TINTA
	HOGAR	PLATO
	PAPELERIA	PLUMA
	FERRETERIA	TINTA

III).-

SURTIDO	ARTI	PROVEEDOR
	PLUMA	PARKER
	LAPIZ	DIXON
	TINTA	PARKER
	PERFUME	AVON
	TINTA	DIXON
	PLATO	CIPSAWARE
	LABIAL	AVON
	PLATO	DIXON
	PLUMA	AVON
	LAPIZ	PARKER

IV).-

TIPO	ARTI	COLOR	TAMAÑO
	PLATO	BLANCO	M
	LABIAL	ROJO	G
	PERFUME	BLANCO	G
	PLUMA	VERDE	C
	LAPIZ	AZUL	M
	TINTA	VERDE	G
	TINTA	AZUL	C
	LAPIZ	ROJO	G
	LAPIZ	AZUL	G

Las respuestas de las solicitudes mostradas son:

1).-

TIPO	COLOR
	BLANCO
	ROJO
	VERDE
	AZUL

2).-

TIPO	ARTI	COLOR	TAMAÑO
	PLATO	BLANCO	M
	LABIAL	ROJO	G
	.	.	.
	:	:	:

3).-

EMP	NOMBRE	SAL	DEPT

4).-

TIPO	ARTI
	PLUMA
	TINTA

5).-

EMP	NOMBRE	SAL
	CISNEROS	16000

6).-

VENTAS	DEPT
	PAPELERIA
	COSMETICOS
	HOGAR

7).-

VENTAS	DEPT
	PAPELERIA
	HOGAR
	COSMETICOS
	JUGUETERIA
	FERRETERIA

8).-

TABLA RESULTA	COSAS	XXX
	PAPELERIA	DIXON
	PAPELERIA	CIPSAWARE
	HOGAR	PARKER
	:	:

9).-

EMP	NOMBRE
	PEREZ
	RANGEL
	OLGUIN

10).-

EMP	NOMBRE
	SANCHEZ
	JUNCUA

11).-

EMP	NOMBRE
	MACHUCA
	CISNEROS

12).-

EMP	NOMBRE CNT
	10

13).-

EMP	NOMBRE SUM
	21000

14).-

EMP	SAL SUM	DEPT
	16000	HOGAR
	21000	JUGUETERIA
	23000	COSMETICOS
	24000	PAPELERIA

15).-

VENTAS	DEPT
	PAPELERIA

16).-

VENTAS	DEPT
	PAPELERIA
	JUGUETERIA

17).-

VENTAS	DEPT

APENDICE II

CARACTERISTICAS DE USO DEL LENGUAJE.

La Implementación del Sistema del Lenguaje de Consulta de Bases de Datos que comprende este trabajo, presenta algunas características para la generación de Solicitudes a la base. Estas deben de ser consideradas para obtener una operación correcta del sistema. Estas peculiaridades son:

1.- Cualquier operador de los contemplados en el sistema (I., E., Todo., etc.), debe ir separado de algun otro, que aparezca en la misma entrada, o de un elemento (ejemplo o constante), por un espacio en blanco al definirse la solicitud. Por ejemplo:

"I. E. Parker"

Espacios en blanco

2.- Los Operadores Relacionales que se empleen, deberán estár entre el operador "C." o "E." y el Elemento (Constante o Ejemplo) que corresponda. Un espacio en blanco se utilizará para separar al operador relacional del operador "C." o "E." y del elemento, esto es:

"C. > 10000"

Espacio en blanco
Operador Relacional

3.- Un Nombre que sea utilizado como un Elemento Ejemplo no podrá usarse como un Elemento Constante o viceversa.

4.- En una Linea no podrá aparecer una entrada con un Elemento Cosntante y otra con un Elemento Ejemplo que ligue a esta con la Caja de Condición en la que se definan otras constantes. Esto es, una solicitud como la siguiente no es válida:

```
Emp (Nombre: I., Sal: E. S1, Dept: C. Zapateria)
Caja Condicion (E. S1 = (10000 ; 13000 ; 15000))
```

5.- Un Elemento Ejemplo unicamente se podrá utilizar para encadenar un par de entradas de lineas diferentes. Esto es, no se podra terner una solicitud como la que sigue:

```
Emp (Nombre: C. Torres, Dept: E. Ejemplo1)
Emp (Sal: > 15000, Dept: E. Ejemplo1)
Ventas (Dept: E. Ejemplo1, Arti: C. Pluma)
```

6.- Para que dos campos de relaciones diferentes puedan ser 'Ligados' (por las operaciones de 'Junta', 'Junta de Grupo' o 'Relación de Equivalencia') estos deberán caer en el mismo dominio.

7.- Una Relación podrá agruparse por un solo campo de la misma. Esto es:

Tipo (Color: I. A. E. Blanco, Tamaño: A. E. Mediano) --> No

Emp (Dept: I. A. E. Jugueteria) --> Si

8.- En una Tabla el Operador 'Todo.' solo puede aparecer en un campo (entrada) de la misma. Esto es:

Emp (Nombre: I. Todo. E. Juan, Dept: Todo. E. Nada) --> No
Emp (Nombre: I. SUM. Todo. E. Jose) --> Si

9.- Cuando se utilicen Funciones Pre-fabricadas, ninguna de las lineas de la solicitud deberán estar marcadas como 'No unibles'. Esto es, no es válida una solicitud como la siguiente:

~ Surtido (Arti: E. Tinta, Proveedor: C. Parker)
Ventas (Dept: I., Arti: SUM. Todo. E. Tinta)

10.- En una solicitud no podrán aparecer más de una Función Pre-fabricada.

11.- Solo se podrá tener una Linea marcada como 'No unible' en la solicitud.

12.- Cuando se tenga una Linea marcada solo se podrán especificar operaciones de Selección y de Junta en la solicitud, y no se podrán emplear 'Tablas de Resultados'.

14.- Al tener en la solicitud alguna Tabla marcada no se podrán realizar operaciones de grupo.

15.- Cuando se llegue a requerir una solicitud en la que alguna de las Tablas deba de marcarse, esta deberá de definirse antes que cualquier otra. Esto es:

Ventas (Dept: I., Arti: E. Tinta) --> Incorrecto
~ Surtido (Arti: E. Tinta, Proveedor: C. Parker)

~ Surtido (Arti: E. Tinta, Proveedor: C. Parker)
Ventas (Dept: I., Arti: E. Tinta) --> Correcto

16.- Las Tablas que tienen asociada una Relación de la base, deberán: contener en cualquiera de sus entradas al operador "I."; o tener alguna de sus entradas ligada a una Tabla que contenga al operador mencionado, con la exepción de la Tabla de Resultados y la Caja de Condición. Por ejemplo:

Ventas (Dept: E. Juguete, Arti: E. Tinta)
Surtido (Arti: E. Tinta, Proveedor: E. IBM)
Tabla Resulta (Cosas: I. E. Juguete, XXX: I. E. IBM)

17.- Cuando se utilice una Tabla de Resultados no se podrá definir Funciones Pre-fabricadas ni Agrupamientos, solo Selecciones y Juntas.

18.- Cuando se vayan a 'encadenar' dos Tablas, en donde ninguna de las dos está marcada como no unible, y en alguna de ellas aparezca el operador "I.", al momento de dar la solicitud se deberá de definir primero a la linea con tal operador. Por ejemplo:

Ventas (Dept: C. Jugueteria, Arti: E. Nuez) --> Incorrecto
Tipo (Arti: I. E. Nuez, Color: C. Verde)

Tipo (Arti: I. E. Nuez, Color: C. Verde) --> Correcto
Ventas (Dept: C. Jugueteria, Arti: E. Nuez)

19.- Solo se podran definir a lo más dos constantes en entradas diferentes para una misma tabla. Esto es:

Type (Arti: C. Plato, Color: C. Rojo, Tamaño: C. M) --> No

Type (Arti: C. Plato, Tamaño: C. G) --> Si
Type (Arti: C. Plato) --> Si

20.- Para una misma entrada solo se podrán definir hasta tres condiciones en la Caja de Condición. Esto significa que:

Emp (Nombre: I., Sal: E. S1) --> Incorrecto
Caja Condición (E. S1 (10000 ¦ 12000 ¦ 13000 ¦ 15000))

Emp (Nombre: I., Sal: E. S1) --> Correcto
Caja Condición (E. S1 (10000 ¦ 13000 ¦ 15000))

Nota.- Los ejemplos que aparecen en este apéndice son meramente ilustrativos y no necesariamente corresponde a una solicitud real.

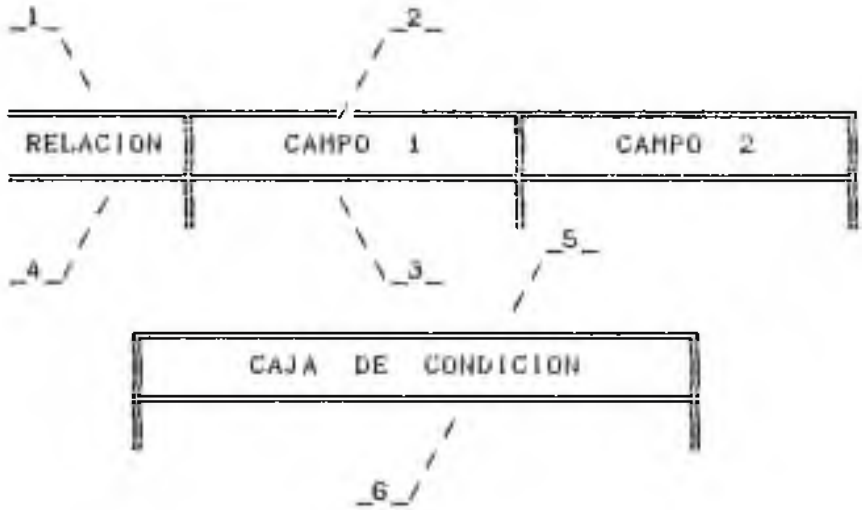
APENDICE III

SINTAXIS DE QUERY-BY-EXAMPLE

A continuación se da una descripción de la sintaxis, de la definición original, de Query-by-Example. Esta solo comprende la definición del lenguaje que se considera en este trabajo.

Dentro de ésta sintáxis los "( )" son usados para indicar entradas opcionales. Partes separadas por "!" representan partes alternativas. Notese que los paréntesis cuadrados y una barra vertical son parte del lenguaje.

Las entradas que se pueden especificar en un esqueleto de Relacion o Tabla de Resultados y una Caja de Condición son:



Donde:

(1).- Entrada-nombre-tabla.
(2).- Entrada-nombre-columna.
(3).- Entrada-columna.
(4).- Entrada-atributo-linea.
(5).- Nombre-Caja-Condición.
(6).- Entrada-Caja-Condición.

Abreviaciones:

el: elemento.
ee: elemento ejemplo.
ec: elemento constante.

Sintaxis:

1).- <el>::= <ee> ! <ec>
2).- <ee>::= [<op-rela>] <id> ! [<op-rela>] <id>
3).- <ec>::= [<op-rela>] <id>
4).- <Entrada-nombre-tabla>::= [<ec>]
5).- <Entrada-nombre-columan>::= [<ec>]
6).- <Entrada-columna>]::= [<expr1>]
7).- <expr1>::= [<op1>] [<el>] ! [<op1>] <fun-conj> <conj>
8).- <op1>::= P.
9).- <op-rela>::= ^ ! = ! > ! >= ! < ! <=
10).- <fun-conj>::= CNT. ! SUM. ! AVE. ! MAX. ! MIN.
11).- <conj>::= <conj-simple> ! <con-brack>
12).- <conj-simple>::= All. [<ee>]
13).- <conj-brack>::= [All. <el> (*)]
14).- Entrada-atributo-linea::= ~
15).- Nombre-Caja-Condición::= COND. BOX.
16).- Entrada-Caja-Condición::= <expr2> ! <expr4>
17).- <expr2>::= <ee> = <expr3>
18).- <expr3>::= [<op-rela>] <ec> ! [<op-rela>] <ec> <expr3>
19).- <expr4>::= <fun-conj><conj-simple><op-rela><ec>

APENDICE IV

REQUERIMIENTOS MINIMOS DE OPERACION.

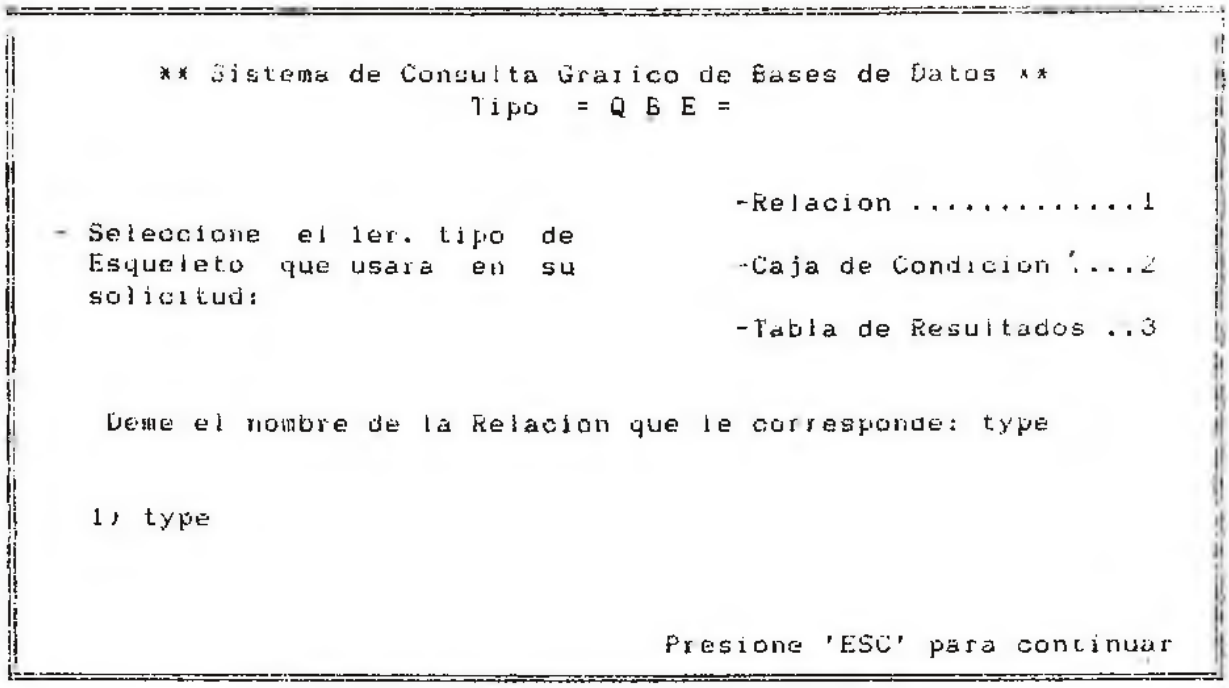
Los requerimientos mínimos de operación del Sistema de consulta de Base de Datos tipo Query-by-Example son:

- Computadora PC o compatible.
- 512 Kbytes de memoria RAM.
- 2 Unidades de disco flexible de 360 Kbytes c/u.
- Reloj de 4.77 Mhz.
- Sistemema Operativo MS-DOS.
- BIOS 3.1.

APENDICE V

EJEMPLOS DE CORRIDAS DEL SISTEMA.

A continuación se presentan los desplegados, resultados y tiempos de ejecución de distintas etapas del sistema, generados por la realización de consultas del tipo de las mostradas en los ejemplos 2, 3, 4, 6, 10, 11 y 15 del capitulo II, a una base de datos que se generó para este fin.



type	ITEM	COLOR	SIZE
	1.	1.	1.

```
    1  Dish      White  M
    2  Ink       Blue   S
    3  Ink       Green  L
    4  Lipstick  Red    L
    5  Pen       Green  S
    6  Pencil    Blue   L
    7  Pencil    Blue   M
    8  Pencil    Red    L
    9  Perfume   White  L
ress any key to continue...
```

```
eneracion del Arbol:              Inicio: 04:22:40    Finalizacion: 04:22:40
ecorrdio y manejo de la B. D.:    Inicio: 04:22:40    Finalizacion: 04:23:03
```

```
        ** Sistema de Consulta Grafico de Bases de Datos **
                        Tipo  = Q B E =


                                              -Relacion .............1
  - Seleccione  el  1er. tipo  de
    Esquereto  que usara  en  su              -Caja de Condicion ....2
    solicitud:
                                              -Tabla de Resultados ..3


     Deme el nombre de la Relacion que le corresponde: emp


     1) emp


                                        Presione 'ESC' para continuar
```

emp	NAME	SAL	DEPT
	1.	C. > 10000	C. Toy

```
  Ningun registro recuperado
Press any key to continue...



 Generacion del Arbol:              Inicio: 04:35:31    Finalizacion: 04:35:32
 Recorrdio y manejo de la B. D.:    Inicio: 04:35:32    Finalizacion: 04:35:52
B>
```

** Sistema de Consulta Grafico de Bases de Datos **
Tipo = Q B E =

- Seleccione el ler. tipo de Esqueleto que usara en su solicitud:

-Relacion 1
-Caja de Condicion 2
-Tabla de Resultados ..3

Deme el nombre de la Relacion que le corresponde: sales

1) type
2) sales

Presione 'ESC' para continuar

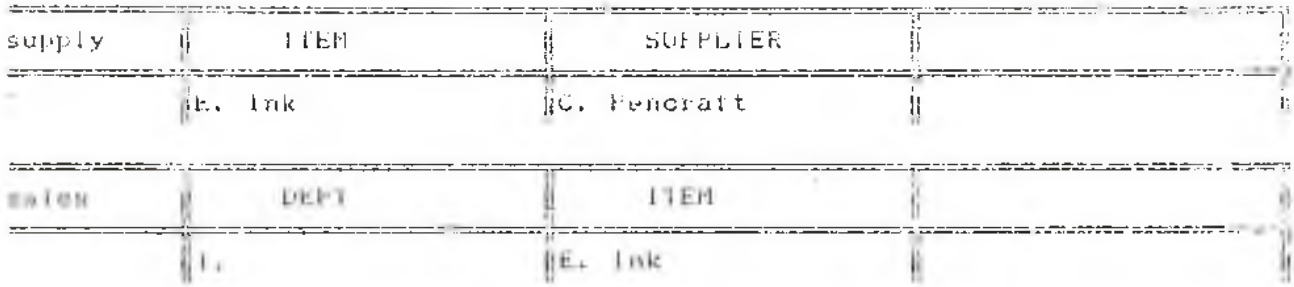
type	ITEM	COLOR	SIZE
	I. E. Nut	C. Green	

sales	DEPT	ITEM	
	C. Toy	E. Nut	

1 Ink
2 Pen
Press any key to continue...

Generacion del Arbol: Inicio: 04:45:59 Finalizacion: 04:46:01
Recorrdio y manejo de la B. D.: Inicio: 04:46:01 Finalizacion: 04:46:32
3>

```
        ** Sistema de Consulta Grafico de Bases de Datos **
                        Tipo  = Q B E =


                                                -Relacion .............1

  - Seleccione  el  1er.  tipo  de
    Esqueleto  que  usara  en  su               -Caja de Condicion ....2
    solicitud:

                                                -Tabla de Resultados ..3


     Deme el nombre de la Relacion que le corresponde: sales


     1) supply
     2) sales


                                        Presione 'EsC' para continuar
```




```
     1  Cosmetics
     2  Household
     3  Stationery
Press any key to continue...
```

```
Generacion del Arbol:              Inicio: 04:52:57    Finalizacion: 04:52:58
Recorrdio y manejo de la B. D.:    Inicio: 04:52:58    Finalizacion: 04:53:39
3>
```

** Sistema de Consulta Grafico de Bases de Datos **
Tipo = Q u E =

-Relacion 1

Selecciono el ter. tipo de esqueleto que usara en su solicitud:

-Caja de Condicion 2

-Tabla de Resultados ...3

Deme el nombre de la Relacion que le corresponde:

1) emp
2) CC

Presione 'ESC' para continuar

emp	NAME	SAL	DEPT
	1.	E. S1	

CAJA DE CONDICION
E. S1 = (> 10000 & < 15000 & = 13000)

1 Lewis
2 Smith
Press any key to continue...

Generacion del Arbol: Inicio: 05:01:44 Finalizacion: 05:01:45
Recorrido y manejo de la B. D.: Inicio: 05:01:45 Finalizacion: 05:02:11

** Sistema de Consulta Grafico de Bases de Datos **
Tipo = Q B E =

Seleccione el ter. tipo de Esqueleto que usara en su solicitud:

-Relacion 1
-Caja de Condicion 2
-Tabla de Resultados ..3

Deme el nombre de la Relacion que le corresponde: emp

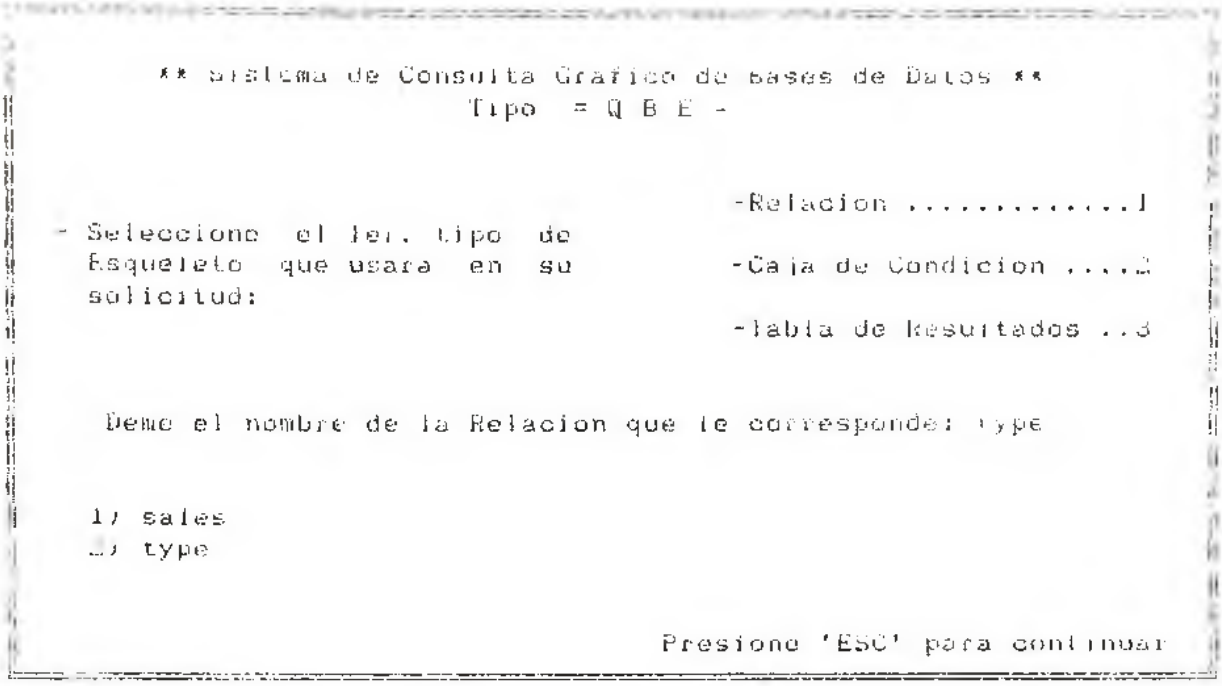
1) emp

Presione 'ESC' para continuar

emp	NAME	SAL	DEPT
	I, CHI, Todo, E, Jo		

1 10
Press any key to continue...

Generacion del Arbol: Inicio: 05:07:50 Finalizacion: 05:07:50
Recorrdio y manejo de la B. D.: Inicio: 05:07:50 Finalizacion: 05:08:09
B>



sales	DEPT	ITEM	
	T. A. E. Toy	Todo. E. Ink, *1	

type	ITEM	COLOR	SIZE
	Todo. E. Ink	C. Green	

```
      4  DEPT        Stationery
      5  DEPT        Toy
Press any key to continue...


 Generacion del Arbol:                Inicio: 05:14:07    Finalizacion: 05:14:08
 Recorrdio y manejo de la B. D.:      Inicio: 05:14:08    Finalizacion: 05:15:10
B>
```

```
***      PROCEDIMIENTOS EN CLIPPER DEL ARCHIVO  UNE_CLIP.ISI    ****

Operacion =
Operando =
Un_arch_in =
N_arch_in =
File =
Ban =  s
Mas_ram =  n
Un_arch_s =
Nom_ar_s =
Arch_out =
Rela_1 = '1
Rela_2 =  2
Rela_3 = '3
Rela_4 =  4
Call Fanta With Rela_1, Rela_2, Rela_3, Rela_4
Cade_1 =  1
Cade_2 = '2
Cade_3 =  3
Cade_4 =  4
Do Nuevo2 With Rela_1, Rela_2, Rela_3, Rela_4, Cade_1, Cade_2, Cade_3, Cade_4
Call Esquelet With Cade_1, Cade_2, Cade_3, Cade_4
Clear
Call Arbol
Mas_ramas =  s
Agrup =  n
Grupo =  n'
Do While Substr(Mas_ramas, 1, 1) =  s
  Call Prim_nodo With Operacion, Operrando, File
  Longi = Len(File)
  Arch_in = Substr(File, 1, Longi)
  Arch_in = Trim(Arch_in)
  Longi = Len(Operrando)
  Do While Substr(Operrando, Longi, 1) = '
    Longi = Longi - 1
  EndDo
  Operandos = Substr(Operrando, 1, Longi)
  Operandos = Trim(Operandos)
  If Substr(Operacion, 1, 1) =  N'
    Mas_ramas =  s '
    Loop
  EndIf
  Do Case
       Case Substr(Operacion, 1, 1) =  S' .And. Substr(Operacion, 2, 1) #  g
          Al = Arch_in +  .dbf'
          If .Not. File(Al)
            All = Arch_in +  2
            Arch_in = Substr(Arch_in, 1, 1) +  2'
            Use &All
            Copy to &Arch_in
          EndIf
          If At( && , Operandos) # 0 .And. Substr(Operandos, 1, 1) #;
             Substr(Operandos, At( && , Operandos) + 3, 1)
```

```
   Op_logico = At( && , Operandos)
   Oper1 = Substr(Operandos, 1, Op_logico - 2)
   Oper2 = Substr(Operandos, Op_logico + 3,;
                  Len(Trim(Operandos)) - Op_logico - 2)
   Do Selecci With Arch_in, Oper1
   Tempo1 = Arch_in +  .dbf'
   Tempo2 = Tempo1
   Rename &Tempo1 to Tempo.dbf
   Tempo1 =  Tempo
   Do Selecci With Tempo1, Oper2
   Tempo1 = Tempo1 + '.dbf
   Rename &Tempo1 to &Tempo2
Else
  If At( && , Operandos) # 0 .Or. At('!! , Operandos) # 0
    Archi = Substr(Arch_in, 1, 2) +  .dbf
    Do Case
         Case At( && , Operandos) # 0
           Op_logico = At( && , Operandos)
           Do While Op_logico # 0
             Oper = Substr(Operandos, 1, Op_logico - 2)
             Operandos = Substr(Operandos, Op_logico + 3,;
                         Len(Trim(Operandos)) - Op_logico - 2)
             Do Selecci With Arch_in, Oper
             Arch_in = Arch_in +  .dbf
             Rename &Arch_in to Tempo.dbf
             Arch_in = 'Tempo
             Op_logico = At( && , Operandos)
           EndDo
           Do Selecci With Arch_in, Operandos
           Arch_in = Arch_in +  .dbf
           Rename &Arch_in to &Archi
           Arch_in = Substr(Archi, 1, Len(Trim(Archi)) - 4)
         Case At( !! , Operandos) # 0
           Archi = Arch_in +  .dbf'
           Conta = 0
           Op_logico = At('!!', Operandos)
           Do While Op_logico # 0
             Oper = Substr(Operandos, 1, Op_logico - 2)
             Operandos = Substr(Operandos, Op_logico + 3,;
                         Len(Trim(Operandos)) - Op_logico - 2)
             Conta = Conta + 1
             If Conta = 1
               Use &Archi
               Copy To T1.dbf
               T1 = 'T1
               Do Selecci With T1, Oper
             Else
               Use &Archi
               Copy To T2.dbf
               T2 = 'T2'
               Do Selecci With T2, Oper
             EndIf
             Op_logico = At('!!', Operandos)
           EndDo
```

```
                Conta = Conta + 1
                If Conta = 2
                  Use &Archi
                  Copy To T2.dbf
                  T2 = 'T2'
                  Do Selecci With T2, Operandos
                Else
                  Use &Archi
                  Copy To T3.dbf
                  T3 = 'T3'
                  Do Selecci With T3, Operandos
                Endif
                Contai = 2
                Close Databases
                Arch_in = Arch_in + '.dbf'
                Erase &Arch_in
                Arch_in = Substr(Arch_in, 1, 2) + '_s'
                Use T1_s
                Copy to &Arch_in
                Use
                Erase T1_s.dbf
                Use &Arch_in
                Do While Contai <= Conta
                  If Contai = 2
                    Archi = 'T2_s'
                  Else
                    Archi = 'T3_s'
                  Endif
                  Append From &Archi
                  Archi = Archi + '.dbf'
                  Erase &Archi
                  Contai = Contai + 1
                EndDo
            EndCase
          Else
            Do Selecci With Arch_in, Operandos
          Endif
        Endif
Case Substr(Operacion, 1, 1) = 'J' .And. Substr(Operacion, 2, 1) # '9'
  Sep = At('-', Arch_in)
  Arch_2 = Substr(Arch_in, Sep + 1, Len(Trim(Arch_in)) - Sep)
  Arch_in = Substr(Arch_in, 1, Sep -1)
   A2 = Arch_2 + '.dbf'
   A1 = Arch_in + '.dbf'
   If .Not. File(A1)
     A11 = Arch_in + '2.dbf'
     Arch_in = Substr(Arch_in, 1, 1) + '2.dbf'
     Use &A11
     Copy to &Arch_in
   Endif
  If .Not. File(A2)
    A11 = Arch_2 + '2.dbf'
    Arch_2 = Substr(Arch_2, 1, 1) + '2.dbf'
    Use &A11
```

```
        Copy to &Arch_2
      Endif
      Do Joins With Arch_in, Arch_2, Operandos
    Case Substr(Operacion, 1, 2) = 'Pu' .And. Grupo = 'n'
      Do Pry_uni With Arch_in, Operandos
    Case Substr(Operacion, 1, 1) = 'A'
      Grupo = 's'
      If Upper(Operandos) # 'NADA'
        Agrup = 's'
      Endif
      Num_arch = 0
       A1 = Arch_in + '.dbf'
       If .Not. File(A1)
         A11 = Arch_in + '2.dbf'
         Arch_in = Substr(Arch_in, 1, 1) + '2.dbf'
         Use &A11
         Copy to &Arch_in
       Endif
      Do Agrupar With Arch_in, Operandos, Num_arch
    Case Substr(Operacion, 1, 2) = 'Re'
      Ar_sj = ' '
      Do Se_Joins With Arch_in, Operandos, Ar_sj
      Do Diferen With Arch_in, Ar_sj
EndCase
If Substr(Operacion, 1, 1) = 'A'
  Un_arch_s = 'n'
  Num_ar_s = Str(Num_arch, 1)
Else
  Num_ar_s = '1'
  Un_arch_s = 's'
Endif
Bandera = 's'
Arch_out = Arch_in
Call Regre_para With Arch_out, Num_ar_s, Un_arch_s
Band2 = 's'
Do While substr(Bandera, 1, 1) = 's' .Or. Band2 = 's'
  Band2 = 'n'
  Call Recorrido With Operacion, Operrando, File, Mas_ram, Ban
  Longi = Len(File)
  Arch_in = Substr(File, 1, Longi)
  Arch_in = Trim(Arch_in)
  Longi = Len(Mas_ram)
  Do While Substr(Mas_ram, Longi, 1) =
    Longi = Longi - 1
  Enddo
  Mas_ramas = Substr(Mas_ram, 1, Longi)
  Mas_ramas = Trim(Mas_ramas)
  Longi = Len(Ban)
  Do While Substr(Ban, Longi, 1) = ' '
    Longi = Longi - 1
  EndDo
  Bandera = Substr(Ban, 1, Longi)
  Bandera = Trim(Bandera)
  Longi = Len(Operrando)
```

```
Do While Substr(Operrando, Longi, 1) =
  Longi = Longi - 1
EndDo
Operandos = Substr(Operrando, 1, Longi)
Operandos = Trim(Operandos)
If Substr(Bandera, 1, 1) = 's
  If Substr(Operacion, 1, 1) = 'N'
    Mas_ramas = 's
    Exit
  EndIf
  Do Case
    Case Substr(Operacion, 1, 1) = 'S' .And. Substr(Operacion, 2, 1) # 'g
      Al = Arch_in + '.dbf'
      If .Not. File(Al)
        All = Arch_in + '2.dbf'
        Arch_in = Substr(Arch_in, 1, 7) + '2.dbf'
        Use &All
        Copy To &Arch_in
      EndIf
      If At('&&' , Operandos) # 0 .And. Substr(Operandos, 1, 1) #;
         Substr(Operandos, At( && , Operandos) + 3, 1)
         Op_logico = At( && , Operandos)
         Oper1 = Substr(Operandos, 1, Op_logico - 2)
         Oper2 = Substr(Operandos, Op_logico + 3,;
                        Len(Trim(Operandos)) - Op_logico - 2)
         Do Selecci With Arch_in, Oper1
         Tempo1 = Arch_in + '.dbf'
         Tempo2 = Tempo1
         Rename &Tempo1 to Tempo.dbf
         Tempo1 = 'Tempo
         Do Selecci With Tempo1, Oper2
         Tempo1 = Tempo1 + '.dbf'
         Rename &Tempo1 to &Tempo2
      Else
        If At( && , Operandos) # 0 .Or. At('||', Operandos) # 0
         Archi = Substr(Arch_in, 1, 2) + '.dbf'
         Do Case
              Case At('&& , Operandos) # 0
                Op_logico = At('&&', Operandos)
                Do While Op_logico # 0
                  Oper = Substr(Operandos, 1, Op_logico - 2)
                  Operandos = Substr(Operandos, Op_logico + 3,;
                              Len(Trim(Operandos)) - Op_logico - 2)
                  Do Selecci With Arch_in, Oper
                  Arch_in = Arch_in + '.dbf'
                  Rename &Arch_in to Tempo.dbf
                  Arch_in = 'Tempo
                  Op_logico = At( && , Operandos)
                EndDo
                Do Selecci With Arch_in, Operandos
                Arch_in = Arch_in + '.dbf'
                Rename &Arch_in to &Archi
                Arch_in = Substr(Archi, 1, Len(Trim(Archi)) - 4)
              Case At('||', Operandos) # 0
```

```
Archi = Arch_in +  .dbf
Conta = 0
Op_logico = At( ||  , Operandos)
Do While Op_logico # 0
  Oper = Substr(Operandos, 1, Op_logico - 2)
  Operados = Substr(Operandos, Op_logico + 3,;
                 Len(Trim(Operandos)) - Op_logico - 2)
  Conta = Conta + 1
  If Conta = 1
    Use &Archi
    Copy To T1.dbf
    T1 = 'T1
    Do Selecci With T1, Oper
  Else
    Use &Archi
    Copy To T2.dbf
    T2 =  T2
    Do Selecci With T2, Oper
  EndIf
  Op_logico = At( ||', Operandos)
EndDo
Conta = Conta + 1
If Conta = 2
  Use &Archi
  Copy To T2.dbf
  T2 = 'T2'
  Do Selecci With T2, Operandos
Else
  Use &Archi
  Copy To T3.dbf
  T3 =  T3'
  Do Selecci With T3, Operandos
EndIf
Contai = 2
Close Databases
Arch_in = Arch_in +  .dbf
Erase &Arch_in
Arch_in = Substr(Arch_in, 1, 2) +  _s'
Use T1_s
Copy to &Arch_in
Use
Erase T1_s.dbf
Use &Arch_in
Do While Contai <= Conta
  If Contai = 2
    Archi =  T2_s'
  Else
    Archi =  T3_s
  EndIf
  Append From &Archi
  Archi = Archi +  .dbf
  Erase &Archi
  Contai = Contai + 1
EndDo
```

```
      EndCase
    Else
      Do Selecci With Arch_in, Operandos
    EndIf
  EndIf
Case Substr(Operacion, 1, 1) = 'J' .And. Substr(Operacion, 2, 1) # 'g'
  Sep = At( - , Arch_in)
  Arch_2 = Substr(Arch_in, Sep + 1, Len(Trim(Arch_in)) - Sep)
  Arch_in = Substr(Arch_in, 1, Sep -1)
  A2 = Arch_2 + '.dbf'
  A1 = Arch_in + '.dbf'
  If .Not. File(A1)
    A11 = Arch_in + '2.dbf'
    Arch_in = Substr(Arch_in, 1, 1) + '2.dbf'
    Use &A11
    Copy To &Arch_in
  EndIf
  If .Not. File(A2)
    A11 = Arch_2 + '2.dbf'
    Arch_2 = Substr(Arch_2, 1, 1) + '2.dbf
    Use &A11
    Copy To &Arch_2
  EndIf
  Do Joins With Arch_in, Arch_2, Operandos
Case Substr(Operacion, 1, 2) = 'Pu' .And. Grupo = 'n'
  Do Pry_uni With Arch_in, Operandos
Case Substr(Operacion, 1, 1) = 'A'
  Grupo = 's'
  If Upper(Operandos) # 'NADA'
    Agrup = 's'
  EndIf
  Num_arch = 0
  A1 = Arch_in + '.dbf'
  If .Not. File(A1)
    A11 = Arch_in + '2.dbf'
    Arch_in = Substr(Arch_in, 1, 1) + '2.dbf
    Use &A11
    Copy To &Arch_in
  EndIf
  Do Agrupar With Arch_in, Operandos, Num_arch
Case Substr(Operacion, 1, 2) = 'Jg'
  Operandos = Trim(Operandos)
  Arch_in = Trim(Arch_in)
  Do Case
    Case At( = , Operandos) # 0
      Oper = '='
    Case At( <', Operandos) # 0
      Oper = '<'
    Otherwise
      Oper = '>'
  EndCase
  Conser = Substr(Operandos, Len(Operandos), 1)
  Operandos = Substr(Operandos, 1, At(Conser, Operandos) - 2)
  If Conser = 'd'
```

```
    A_conserv = Substr(Arch_in, At('\', Arch_in) + 1,;
                       Len(Arch_in) - At('\', Arch_in))
    Ar_2 = A_conserv
    A_borrar = Substr(Arch_in, 1, At('\', Arch_in) - 1)
    Ar_1 = A_borrar
    Ca_2 = Substr(Operandos, At(Oper, Operandos) + 2,;
                  Len(Operandos) - At(Oper, Operandos) - 1)
    Ca_1 = Substr(Operandos, 1, At(Oper, Operandos) - 2)
  Else
    A_conserv = Substr(Arch_in, 1, At('\', Arch_in) - 1)
    Ar_1 = A_conserv
    A_borrar = Substr(Arch_in, At('\', Arch_in) + 1,;
                       Len(Arch_in) - At('\', Arch_in))
    Ar_2 = A_borrar
    Ca_1 = Substr(Operandos, 1, At(Oper, Operandos) - 2)
    Ca_2 = Substr(Operandos, At(Oper, Operandos) + 2,;
                  Len(Operandos) - At(Oper, Operandos) - 1)
  EndIf
  Use &A_conserv
  Go Bottom
  Num_rec = Recno()
  I = 1
  Do While I <= Num_rec
    Go I
    Archivo = Trim(Nom_arch)
    Verdad = .T.
    If Oper = '='
      If Conser = 'd'
        Do Join_G With Ca_2, Archivo, Ca_1, Ar_1, '<', Verdad
      Else
        Do Join_G With Ca_2, Ar_2, Ca_1, Archivo, '<', Verdad
      EndIf
    Else
      If Conser = 'd'
        Do Join_G With Ca_1, Ar_1, Ca_2, Archivo, Oper, Verdad
      Else
        Do Join_G With Ca_1, Archivo, Ca_2, Ar_2, Oper, Verdad
      EndIf
    EndIf
    Archivo = Archivo + '.dbf'
    Erase &Archivo
    Use &A_conserv
    Go I
    Replace Vive With Verdad
    I = I + 1
  EndDo
  Close Databases
  A_borrar = A_borrar + '.dbf'
  Erase &A_borrar
  Arch_in = A_conserv
Case Substr(Operacion, 1, 2) = 'Re'
  Ar_sj = ''
  Do Se_Joins With Arch_in, Operandos, Ar_sj
  Do Diferen With Arch_in, Ar_sj
```

```
Case Substr(Operacion, 1, 2) = 'Fp'
  Operandos = Trim(Operandos)
  Arch_in = Trim(Arch_in)
  Campo = Substr(Operandos, 1, At(' ', Operandos) - 1)
  Funcion = Substr(Operandos, At('  ', Operandos) + 1,;
            Len(Operandos) - At('  ', Operandos))
  Arch_out = '
  Do Gen_a_fp With Funcion, Arch_out
  Resultado = 0
  If Agrup = 'n'
    Do Case
      Case Funcion = 'SUM'
        Do Suma With Campo, Arch_in, Resultado
      Case Funcion = 'CNT'
        Do Cuenta With Arch_in, Resultado
      Case Funcion = 'PRM'
        Do Promedio With Campo, Arch_in, Resultado
      Case Funcion = 'MAX'
        Do Max_Min With Campo, Arch_in, 'Maximo', Resultado
      Otherwise
        Do Max_Min With Campo, Arch_in, 'Minimo ', Resultado
    EndCase
    Do Fon_resu With Arch_out, Resultado, Arch_in, Campo
    Close Databases
    Arch_in = Arch_in + '.dbf'
    Erase &Arch_in
  Else
    Use &Arch_in
    Go Bottom
    Num_rec = Recno()
    Reg = 1
    Do While Reg <= Num_rec
      Go Reg
      Archivo = Nom_arch
      Do Case
        Case Funcion = 'SUM'
          Do Suma With Campo, Archivo, Resultado
        Case Funcion = 'CNT'
          Do Cuenta With Archivo, Resultado
        Case Funcion = 'PRM'
          Do Promedio With Campo, Archivo, Resultado
        Case Funcion = 'MAX'
          Do Max_Min With Campo, Archivo, 'Maximo ', Resultado
        Otherwise
          Do Max_Min With Campo, Archivo, 'Minimo ', Resultado
      EndCase
      Do Fon_resu With Arch_out, Resultado, Archivo, Campo
      Close Databases
      Archivo = Trim(Archivo) + '.dbf'
      Erase &Archivo
      Use &Arch_in
      Reg = Reg + 1
    EndDo
  EndIf
```

```
        Case Substr(Operacion, 1, 2) = 'Sg
          Operandos = Trim(Operandos)
          Do Case
            Case At( = , Operandos) # 0
              Operan =  =
            Case At( < , Operandos) # 0
              Operan =  <
            Case At('>', Operandos) # 0
              Operan =  >
            Case At('<=', Operandos) # 0
              Operan =  <=
            Case At('>=', Operandos) # 0
              Operan =  >=
            Otherwise
              Operan =  #
          EndCase
          Pos_operan = At(Operan, Operandos)
          Pos_blanc = At(   , Operandos)
          Fun_pre = Substr(Operandos, Pos_blanc + 1,;
                           Pos_operan - Pos_blanc - 2)
          Do Case
            Case Fun_pre =  SUM
              Archivo =  Suma
            Case Fun_pre =  CNT
              Archivo =  Cuenta
            Case Fun_pre =  PRM
              Archivo =  Promedio
            Case Fun_pre = 'MAX'
              Archivo =  Maximo
            Otherwise
              Archivo =  Minimo
          EndCase
          Sub_op = Substr(Operandos, Pos_operan,;
                          Len(Operandos) - Pos_operan + 1)
          Pos_blanc = At(' ', Sub_op)
          Condicion = Substr(Sub_op, Pos_blanc + 1,;
                      Len(Sub_op) - Pos_blanc)
           Do Selec_6 With Archivo, Operan, Condicion, Arch_in
        Case Substr(Operacion, 1, 2) =  Pr
          Do Provecc With Operandos, Arch_in
      EndCase
      Arch_out = Arch_in
      Call Regre_para With Arch_out, '1 ,  s'
      Clear
    EndIf
  EndDo
EndDo
```

```
Procedure Selecci
Parameters Archivo, Operandos

Op_rela =
Constante =
Campo =
Do Case
     Case AT( = , Operandos) # 0
          Op_rela = =
     Case At( <', Operandos) # 0
          Op_rela = <'
     Case At( >', Operandos) # 0
          Op_rela = >
     Case At( ' , Operandos) # 0
          Op_rela = "'
EndCase
Constante = Substr(Operandos, At(Op_rela, Operandos) + 2
                   ,Len(Trim(Operandos)) - At(Op_rela, Operandos) - 1)
Campo = Substr(Operandos, 1, At(Op_rela, Operandos) - 2)
Use &Archivo
Copy to Estruc Structure Extended
Select 2
Use Estruc
Go top
Do While Trim(Field_name) # Upper(Trim(Campo))
  Skip
EndDo
Tipo_var = Field_type
Select 1
Archivo2 = Substr(Archivo, 1, 2) +  _s
Do Case
     Case Tipo_var =  N' .OR. Tipo_var =  n
       Do Case
            Case Op_rela =  ='
              Copy to &Archivo2 For &Campo = Val(Constante)
            Case Op_rela = '>
              Copy to &Archivo2 For &Campo > Val(Constante)
            Case Op_rela = '<'
              Copy to &Archivo2 For &Campo < Val(Constante)
            Otherwise
              Copy to &Archivo2 For &Campo # Val(Constante)
       EndCase
     Case Tipo_var =  C' .OR. Tipo_var =  c
       Constantef = "'" + Constante + "'"
       Do Case
            Case Op_rela = '='
              Copy to &Archivo2 For &Campo = &Constantef
            Case Op_rela = '>'
              Copy to &Archivo2 For &Campo > &Constantef
            Case Op_rela = '<'
              Copy to &Archivo2 For &Campo < &Constantef
            Otherwise
              Copy to &Archivo2 For &Campo # &Constantef
       EndCase
```

```
EndCase
Close databases
Archivo = Archivo + .dbf
Erase &Archivo
Archivo = Archivo2
Return
```

```
Procedure Joins
Parameters Archivo1, Archivo2, Operandos

Campo1 =
Campo2 =
Op_rela =
Do Case
     Case At( = , Operandos) # 0
          Op_rela = =
     Case At( < , Operandos) # 0
          Op_rela = <
     Case At( > , Operandos) # 0
          Op_rela = >
     Case At( # , Operandos) # 0
          Op_rela = #
EndCase
Campo1 = Substr(Operandos, 1, At(Op_rela, Operandos) - 2)
Campo2 = Substr(Operandos, At(Op_rela, Operandos) + 2,;
                 Len(Trim(Operandos)) - At(Op_rela, Operandos) - 1)
Archivo = Substr(Archivo1, 1, 2) +  _J
Use &Archivo2
Select 2
Use &Archivo1 Alias Archi
Select 1
Do Case
     Case Op_rela =  =
       Join With Archi to &Archivo For &Campo2 = Archi -> &Campo1
     Case Op_rela =  >
       Join With Archi to &Archivo For &Campo2 < Archi->&Campo1
     Case Op_rela =  <
       Join With Archi to &Archivo For &Campo2 > Archi->&Campo1
     Otherwise
       Join With Archi to &Archivo For &Campo2 # Archi->&Campo1
EndCase
Close Databases
Archivo1 = Archivo1 +  .dbf
Archivo2 = Archivo2 +  .dbf
Erase &Archivo1
Erase &Archivo2
Archivo1 = Archivo
Return
```

```
Procedure Pry_uni
Parameters Archivo, Cam

Campos = Trim(Cam)
Do While At(    , Campos) # 0
  Pos = At('   , Campos)
  Campos = Substr(Campos, 1, Pos - 1) +   +  +;
           Substr(Campos, Pos + 1, Len(Campos) - Pos)
EndDo
Use &Archivo
If Lastrec() < 1
  Return
EndIf
Copy to Estruc Structure Extended
Select 2
Use Estruc
Pos = 1
Dif_tipos =  n
Do While At( + , Substr(Campos, Pos, Len(Campos) - Pos)) # 0
  Pos2 = At( + , Substr(Campos, Pos, Len(Campos) - Pos)) + Pos
  Campo = Substr(Campos, Pos, Pos2 - Pos - 1)
  Go top
  Do While Trim(Field_name) # Upper(Trim(Campo))
    Skip
  EndDo
  If Field_type =  N
    If Pos <= 1
      Campos =  Str(  + Campo +  )  + Substr(Campos, Pos2 + 1,;
                Len(Trim(Campos)) - Pos2 + 2)
    Else
      Campos = Substr(Campos, 1, Pos - 1) +  Str(' + Campo +  )  +;
               Substr(Campos, Pos2 - 1, Len(Trim(Campos)) - Pos2 + 2)
    EndIf
    Pos2 = Pos2 + 5
    Dif_tipos =  s
  EndIf
  Pos = Pos2
EndDo
Campo = Substr(Campos, Pos, Len(Trim(Campos)) - Pos + 1)
Go top
Do While Trim(Field_name) # Upper(Trim(Campo))
  Skip
EndDo
If Field_type = 'N  .And. Pos > 1
  Campos = Substr(Campos, 1, Pos -1) +  Str( + Campo + ')
  Dif_tipos =  s
EndIf
Select 1
Set Unique On
Archivo_P = Substr(Archivo, 1, 2) +  _P
Archivo_U = Substr(Archivo, 1, 2) +  _U
If At( + , Campos) # 0 .OR. At( ( , Campos) # 0
     Do Al_cam_P With Campos
EndIf
```

```
Cam_aux = Campos
Camp1 =
Camp2 =
Camp3 =
Camp4 =
Cuent_camp = 1
Campos = Trim(Campos)
Do While At( , , Campos) # 0
  Pos_coma = At( , , Campos)
  Do Case
    Case Cuent_camp = 1
      Camp1 = Substr(Campos, 1, Pos_coma - 1)
    Case Cuent_camp = 2
      Camp2 = Substr(Campos, 1, Pos_coma - 1)
    Case Cuent_camp = 3
      Camp3 = Substr(Campos, 1, Pos_coma - 1)
    Otherwise
      Camp4 = Substr(Campos, 1, Pos_coma - 1)
  EndCase
  Cuent_camp = Cuent_camp + 1
  Campos = Substr(Campos, Pos_coma + 2, Len(Campos) - Pos_coma - 1)
EndDo
Do Case
  Case Cuent_camp = 1
    Camp1 = Campos
  Case Cuent_camp = 2
    Camp2 = Campos
  Case Cuent_camp = 3
    Camp3 = Campos
  Otherwise
    Camp4 = Campos
EndCase
Do Case
  Case Cuent_camp = 1
    Sort On &Camp1 to &Archivo_U
  Case Cuent_camp = 2
    Sort On &Camp1 , &Camp2 to &Archivo_U
  Case Cuent_camp = 3
    Sort On &Camp1 , &Camp2 , &Camp3 to &Archivo_U
  Otherwise
    Sort On &Camp1, &Camp2, &Camp3, &Camp4 to &Archivo_U
EndCase
Do Sel_unica With Cam_aux, Archivo_U
Use &Archivo_U
Do Case
  Case Cuent_camp = 1
    Copy to &Archivo_P Fields &Camp1
  Case Cuent_camp = 2
    Copy to &Archivo_P Fields &Camp1 , &Camp2
  Case Cuent_camp = 3
    Copy to &Archivo_P Fields &Camp1 , &Camp2 , &Camp3
  Otherwise
    Copy to &Archivo_P Fields &Camp1, &Camp2, &Camp3, &Camp4
EndCase
```

```
Close Databases
Archivo = Archivo +  .dbf
Archivo_U = Archivo_U +  .dbf
Erase &Archivo
Erase &Archivo_U
Erase Estruc.dbf
Archivo = Archivo_P
Return

* ------------------------------------------------------------

Procedure Al_cam_p
Parameters Campos

C =
CC = ' '
If At( + , Campos) # 0
     Do While At( + , Campos) # 0
          Do Cuer_Al With Campos, C, CC,  ,
     EndDo
Endif
Campos = Campos +  +'
Do Cuer_Al With Campos, C, CC, '
Campos = Substr(CC, 2, Len(CC) - 1)
Return

* ------------------------------------------------------------

Procedure Cuer_Al
Parameters Campos, C, CC, Coma

k = At( + , Campos)
C = C + Substr(Campos, 1, k-1)
C = Substr(C, 2, Len(C)-1)
If At( ( , C) # 0
     i = At( ( , C) + 1
     j = At( ) , C)
     CC = CC + Substr(C, i, j - i) + Coma
Else
     CC = CC + C + Coma
Endif
C = ' '
Campos = Substr(Campos, k + 1, Len(Campos) - k)
Return

* ------------------------------------------------------------

Procedure Sel_unica
Parameters Campos, Archivo

Camp1 = '
Camp2 =
Camp3 = '
Camp4 =
```

```
Cuent_camp = 1
Campos = Trim(Campos)
Do While At( , , Campos) # 0
  Pos_coma = At( , , Campos)
  Do Case
    Case Cuent_camp = 1
      Camp1 = Substr(Campos, 1, Pos_coma - 1)
    Case Cuent_camp = 2
      Camp2 = Substr(Campos, 1, Pos_coma - 1)
    Case Cuent_camp = 3
      Camp3 = Substr(Campos, 1, Pos_coma - 1)
    Otherwise
      Camp4 = Substr(Campos, 1, Pos_coma - 1)
  EndCase
  Cuent_camp = Cuent_camp + 1
  Campos = Substr(Campos, Pos_coma + 2, Len(Campos) - Pos_coma - 1)
EndDo
Do Case
  Case Cuent_camp = 1
    Camp1 = Campos
  Case Cuent_camp = 2
    Camp2 = Campos
  Case Cuent_camp = 3
    Camp3 = Campos
  Otherwise
    Camp4 = Campos
EndCase
Use &Archivo
Go top
Do While .Not. Eof()
  Cuenta = 1
  Do While Cuenta <= Cuent_camp
    Do Case
      Case Cuenta = 1
        Field1 = &Camp1
      Case Cuenta = 2
        Field2 = &Camp2
      Case Cuenta = 3
        Field3 = &Camp3
      Otherwise
        Field4 = &Camp4
    EndCase
    Cuenta = Cuenta + 1
  EndDo
  Igual =  's'
  Do While Igual = 's'
    Skip
    Cuenta = 1
    Do While Cuenta <= Cuent_camp
      Do Case
        Case Cuenta = 1
          If Field1 # &Camp1
            Igual =  'n'
          Endif
```

```
        Case Cuenta = 2
          If Field2 # &Camp2
            Igual =  n
          EndIf
        Case Cuenta = 3
          If Field3 # &Camp3
            Igual =  n
          EndIf
        Otherwise
          If Field4 # &Camp4
            Igual =  n
          EndIf
      EndCase
      If Igual =  n
        Exit
      Else
        Cuenta = Cuenta + 1
      EndIf
    EndDo
    If Igual =  s
      Delete
    EndIf
  EndDo
EndDo
Pack
Use
Return
```

```
Procedure Agrupar
Parameters Archivo, Campo, Num_Arch

Archivo_G = Substr(Archivo, 1, 2) +  _So
If Upper(Campo) # 'NADA
  Use Structur
  Go Top
  Delete All
  Pack
  Go Bottom
  I = 1
  Do While I <= 4
    Append Blank
    Do Case
      Case I = 1
        Replace Field_Name With  Num_arch , Field_Type With  Character ,;
                Field_Len With 10
      Case I = 2
        Replace Field_Name With  Cam_agru , Field_Type With  Character ,;
                Field_Len With 10
      Case I = 3
        Replace Field_Name With  Contenid , Field_Type With  Character ,;
                Field_Len With 10
      Otherwise
        Replace Field_Name With  Vive , Field_Type With  Character ,;
                Field_Len With 1
    EndCase
    I = I + 1
  EndDo
  Use &Archivo
  Sort On &Campo /A To &Archivo_G
  Arch_temp = Archivo
  Archivo = Substr(Archivo, 1, 2)
  Close Databases
  Create &Archivo From Structur
  Use &Archivo_G
  Go top
  Select 2
  Use &Archivo
  Go top
  Select 1
  J = 1
  I = 1
  Do While .Not. Eof()
    Variable = &Campo
    Archivo_sal = Archivo + Str(I,2)
    Do While At('  , Archivo_sal) # 0
      K = At(   , Archivo_sal)
      Archivo_sal = Substr(Archivo_sal, 1, K - 1) +  0  +;
                    Substr(Archivo_sal, K + 1, Len(Archivo_sal) - K)
    EndDo
    Copy to &Archivo_sal For &Campo = Variable
    Select 2
    Append Blank
```

```
    Replace Nom_arch With Archivo_sal, Cam_agru With Campo,;
            Contenid With Variable, Vive With  V
    Select 1
    I = I + 1
    Go J
    Do While &Campo <= Variable .And. .Not. Eof()
      Skip
      J = J + 1
    EndDo
  EndDo
  Num_Arch = I - 1
  Archivo_G = Archivo_G +  .dbf
  Arch_temp = Arch_temp +  .dbf
  Close Databases
  Erase &Archivo_G
  Erase &Arch_temp
Else
  Num_Arch = 1
Endif
Return
```

```
Procedure Join_5
Parameters Campo1, Archivo1, Campo2, Archivo2, Operando, Verdad

Archivo1_I = Archivo1 +  _I
Archivo2_I = Archivo2 +  _I
Set Unique On
Use &Archivo1 Alias Archiv1
Index On &Campo1 to &Archivo1_I
Select 2
Use &Archivo2 Alias Archiv2
Index On &Campo2 to &Archivo2_I
Verdad =  V
Select Archiv1
Num_rec1 = Lastrec()
Select 2
Num_rec2 = Lastrec()
Select Archiv1
If Operando =  =
  If Num_rec1 # Num_rec2
    Verdad =  F
  Else
    Do While .Not. Eof()
      If &Campo1 = Archiv2 -> &Campo2
        Skip
        Select Archiv2
        Skip
        If Eof()
          Verdad = F
          Select Archiv1
          If Eof()
            Verdad =  V
          EndIf
          Exit
        EndIf
        Select Archiv1
        Loop
      Else
        Verdad = F
        Exit
      EndIf
    EndDo
  EndIf
Else
  If Num_rec1 >= Num_rec2
    Verdad =  F
  Else
    Do While .Not. Eof()
      If &Campo1 > Archiv2 -> &Campo2
        Select Archiv2
        Skip
        If Eof()
          Verdad = F
          Exit
        EndIf
```

```
        Select Archiv1
      Else
        If &Campo1 = Archiv2 -> &Campo2
          Skip
          Select Archiv2
          Skip
          If Eof()
            Select Archiv1
            If .Not. Eof()
              Verdad = 'F
              Exit
            EndIf
          EndIf
        Select Archiv1
         Else
          Verdad =  F
          Exit
        EndIf
      EndIf
    EndDo
  EndIf
EndIf
Close Databases
Close Index
Archivo1_I = Archivo1_I +  .ntx
Archivo2_I = Archivo2_I +  .ntx
Erase &Archivo1_I
Erase &Archivo2_I
Return
```

```
Procedure Se_Joins
Parameters Arch_in, Operandos, Archivo

Operandos = Trim(Operandos)
Arch_in = Trim(Arch_in)
Do Case
  Case At('= , Operandos) # 0
    Oper =  ='
  Case At( < , Operandos) # 0
    Oper = '<'
  Otherwise
    Oper = '>
EndCase
Conser = Substr(Operandos, Len(Operandos), 1)
Operandos = Substr(Operandos, 1, At(Conser, Operandos) - 2)
If Conser = 'd
  A_conserv = Substr(Arch_in, At( - , Arch_in) + 1,;
                     Len(Arch_in) - At( - , Arch_in))
  A_borrar = Substr(Arch_in, 1, At( - , Arch_in) - 1)
  Campo1 = Substr(Operandos, At(Oper, Operandos) + 2,;
                  Len(Operandos) - At(Oper, Operandos) - 1)
  Campo2 = Substr(Operandos, 1, At(Oper, Operandos) - 2)
Else
  A_conserv = Substr(Arch_in, 1, At( - , Arch_in) - 1)
  A_borrar = Substr(Arch_in, At( - , Arch_in) + 1,;
                     Len(Arch_in) - At( - , Arch_in))
  Campo1 = Substr(Operandos, 1, At(Oper, Operandos) - 2)
  Campo2 = Substr(Operandos, At(Oper, Operandos) + 2,;
                  Len(Operandos) - At(Oper, Operandos) -1)
EndIf
Archivo = Substr(A_conserv, 1, 2) +  _sj
Use &A_conserv
Cadena =
Do Campos_sj With A_conserv, Cadena
Camp1 =
Camp2 =
Camp3 =
Num_camp = 1
Separad =  ,
Do Sep_camp With Cadena, Camp1, Camp2, Camp3, Separad, Num_camp
Use &A_conserv
Select 2
Use &A_borrar
Select 1
Do Case
     Case Oper =  ='
       Do Case
         Case Num_camp = 1
           Join With B to &Archivo For &Campo1 = B -> &Campo2;
                        Fields &Camp1
         Case Num_camp = 2
           Join With B to &Archivo For &Campo1 = B -> &Campo2;
                        Fields &Camp1, &Camp2
         Otherwise
```

```
            Join With B to &Archivo For &Campo1 = B -> &Campo2;
                          Fields &Camp1, &Camp2, &Camp3
        EndCase
      Case Oper = '>'
        Do Case
          Case Num_camp = 1
            Join With B to &Archivo For &Campo1 > B -> &Campo2;
                          Fields &Camp1
          Case Num_camp = 2
            Join With B to &Archivo For &Campo1 > B -> &Campo2;
                          Fields &Camp1, &Camp2
          Otherwise
            Join With B to &Archivo For &Campo1 > B -> &Campo2;
                          Fields &Camp1, &Camp2, &Camp3
        EndCase
      Case Oper = '<'
        Do Case
          Case Num_camp = 1
            Join With B to &Archivo For &Campo1 < B -> &Campo2;
                          Fields &Camp1
          Case Num_camp = 2
            Join With B to &Archivo For &Campo1 < B -> &Campo2;
                          Fields &Camp1, &Camp2
          Otherwise
            Join With B to &Archivo For &Campo1 < B -> &Campo2;
                          Fields &Camp1, &Camp2, &Camp3
        EndCase
      Otherwise
        Do Case
          Case Num_camp = 1
            Join With B to &Archivo For &Campo1 # B -> &Campo2;
                          Fields &Camp1
          Case Num_camp = 2
            Join With B to &Archivo For &Campo1 # B -> &Campo2;
                          Fields &Camp1, &Camp2
          Otherwise
            Join With B to &Archivo For &Campo1 # B -> &Campo2;
                          Fields &Camp1, &Camp2, &Camp3
        EndCase
EndCase
Close Databases
A_borrar = A_borrar + '.dbf'
Erase &A_borrar
Arch_in = A_conserv
Return

* ------------------------------------------------------------

Procedure Campos_sj
Parameters Archivo, Cadena

Use &Archivo
Copy To mA_sj Structure Extended
Use mA_sj
```

```
Go 1
Do While .Not. Eof()
     Cadena = Cadena +  ,  + Trim(Field_name)
     Skip
EndDo
Cadena = Substr(Cadena, 3)
Close Databases
Erase AA_sj.dbf
Return

* ------------------------------------------------------------------

Procedure Sep_camp
Parameters Campos, Camp1, Camp2, Camp3, Separad, Num_camp

Camp1 =
Camp2 =
Camp3 =
Campos = Trim(Campos)
Do While At(Separad, Campos) # 0
  Pos_sep = At(Separad, Campos)
  Do Case
    Case Num_camp = 1
      Camp1 = Substr(Campos, 1, Pos_sep - 1)
    Otherwise
      Camp2 = Substr(Campos, 1, Pos_sep - 1)
  EndCase
  Num_camp = Num_camp + 1
  Campos = Substr(Campos, Pos_sep + 1, Len(Campos) - Pos_sep)
EndDo
If Num_camp = 2
  Camp2 = Campos
Else
  Camp3 = Campos
EndIf
Return
```

```
Procedure Diferen
Parameters Archivo1, Archivo2

Archivo1_S = Substr(Archivo1, 1, 2) +  Su
Archivo2_1 = Archivo2 + '1
Use &Archivo1 Alias Archi1
Copy To Ar_1 Structure Extended
Select 3
Use Ar_1
Go Bottom
Num_Rec = Recno()
Icamp1 =
Icamp2 = ' '
Icamp3 =
Do Encadena With  Ar_1 , Num_Rec, Icamp1, Icamp2, Icamp3
Select 1
Use &Archivo1
Do Case
  Case Num_rec = 1
    Index On &Icamp1 To &Archivo1_S
  Case Num_rec = 2
    Index On &Icamp1 + &Icamp2 To &Archivo1_S
  Otherwise
    Index On &Icamp1 + &Icamp2 + &Icamp3 To &Archivo1_S
EndCase
Select 2
Use &Archivo2 Alias Archi2
Do Case
  Case Num_rec = 1
    Index On &Icamp1 To &Archivo2_1
  Case Num_rec = 2
    Index On &Icamp1 + &Icamp2 To &Archivo2_1
  Otherwise
    Index On &Icamp1 + &Icamp2 + &Icamp3 To &Archivo2_1
EndCase
Fin_2 = .F.
Select 1
Set Index To &Archivo1_S
Go Top
Do While .Not. Eof() .And. .Not. Fin_2
  Do Case
    Case Num_rec = 1
      Cadena2 = &Icamp1
      Cadena3 = Archi2 -> &Icamp1
    Case Num_rec = 2
      Cadena2 = &Icamp1 + &Icamp2
      Cadena3 = Archi2 -> &Icamp1 + Archi2 -> &Icamp2
    Otherwise
      Cadena2 = &Icamp1 + &Icamp2 + &Icamp3
      Cadena3 = Archi2 -> &Icamp1 + Archi2 -> &Icamp2 + Archi2 -> &Icamp3
  EndCase
  If Cadena2 = Cadena3
    Delete
    Skip
```

```
    Select Archi2
    Skip
    Fin_2 = Eof()
    Select 1
  Else
    If Cadena2 < Cadena3
      Skip
    Else
      Select Archi2
      Skip
      Fin_2 = Eof()
      Select 1
    EndIf
  EndIf
EndDo
Pack
Close Databases
Close Index
Archivo2_I = Archivo2_I + .ntx
Archivo1_S = Archivo1_S + .ntx
Archivo2 = Archivo2 + .dbf
Erase &Archivo2
Erase &Archivo2_I
Erase &Archivo1_S
Erase Ar_1.dbf
Return

* ------------------------------------------------------------

Procedure Encadena
Parameters Archivo, Num_rec, Icadena1, Icadena2, Icadena3

Use &Archivo
Go 1
I = 1
Do While I <= Num_rec
  Do Case
    Case Field_type = 'C' .OR. Field_type = 'c'
      Do Case
        Case I = 1
          Icadena1 = Field_name
        Case I = 2
          Icadena2 = Field_name
        Otherwise
          Icadena3 = Field_name
      EndCase
    Case Field_type = 'N' .OR. Field_type = 'n'
      Do Case
        Case I = 1
          Icadena1 = Str(' + Field_name + ')'
        Case I = 2
          Icadena2 = Str(' + Field_name + ')'
        Otherwise
          Icadena3 = Str(' + Field_name + ')'
```

```
      EndCase
    Otherwise
      Do Case
         Case i = 1
           lcadena1 = 'Dtoc(' + Field_name + ')'
         Case i = 2
           lcadena2 = 'Dtoc(' + Field_name + ')'
         Otherwise
           lcadena3 = 'Dtoc(' + Field_name + ')'
      EndCase
  EndCase
  Skip
  i = i + 1
EndDo
Return
```

```
Procedure Gen_e_ip
Parameters Fun_prefa, Operacion

Fun_prefa = Trim(Fun_prefa)
Do Case
     Case Fun_prefa = 'CNT'
        Operacion =  Cuenta
     Case Fun_prefa = 'SUM'
        Operacion =  Suma
     Case Fun_prefa =  'AVG'
        Operacion =  Promedio'
     Case Fun_prefa =  'MAX'
        Operacion =  Maximo
     Otherwise
        Operacion =  Minimo'
EndCase
Use Structur
Go Top
Delete All
Pack
Go Bottom
Append Blank
Temporal = Operacion
Tipo =  Character
I = 1
Do While I <= 4
     If I # 4
          Replace Field_Name With Operacion, Field_Type With  Character , ;
                  Field_Len With 10
     Else
        Replace Field_Name With Operacion, Field_Type With Tipo_Campo,;
             Field_Len With Long_Campo
     Endif
     I = I + 1
     Do Case
        Case I = 2
             Operacion =  Arch_aso
           Append Blank
           Case I = 3
              Operacion = 'Cam_fun'
                 Append Blank
        Case I = 4
           Operacion =  Imprimir'
           Tipo_Campo =  Logical'
           Long_Campo = 3
           Append Blank
     EndCase
EndDo
Operacion = Temporal
Close Databases
Create &Operacion From Structur
Close Databases
Return
```

```
* ------------------------------------------------------------

Procedure Fon_resu
Parameters Operacion, Resultado, Arch_ope, Cam_ope

Operacion = Trim(Operacion)
Use &Operacion
Append Blank
Go Bottom
Do Case
  Case Operacion =  Suma
    Replace Suma With Str(Resultado)
  Case Operacion =  Cuenta
    Replace Cuenta With Str(Resultado)
  Case Operacion =  Promedio
    Replace Promedio With Str(Resultado)
  Case Operacion =  Maximo
    Replace Maximo With Str(Resultado)
  Otherwise
    Replace Minimo With Str(Resultado)
EndCase
Replace Arch_aso With Arch_ope, Cam_fun With Cam_ope, Imprimir With .f.
Return

* ------------------------------------------------------------

Procedure Suma
Parameters Camp_ope, Arch_in, Suma_t

Use &Arch_in Alias Arch_in
Sum &Camp_ope To Suma_t
Go Top
Return

* ------------------------------------------------------------

Procedure Cuenta
Parameters Arch_in, Cuenta_t

Use &Arch_in Alias Arch_in
Count To Cuenta_t
Go Top
Return

* ------------------------------------------------------------

Procedure Promedio
Parameters Camp_ope, Arch_in, Promedio_t

Use &Arch_in Alias Arch_in
Average &Camp_ope To Promedio_t
Go Top
Return
```

```
* ------------ ----------------------------------------

Procedure Max_Min
Parameters Camp_ope, Arch_in, Arch_out, M

Use &Arch_in Alias Arch_in
If Arch_out = 'Maximo
  Operando = '>'
  M = 0
Else
  Operando = '<'
  M = 999999
EndIf
Go Top
Do While .Not. Eof()
  Do Case
    Case Operando = '>'
      If &Camp_ope > M
        M = &Camp_ope
      EndIf
    Otherwise
      If &Camp_ope < M
        M = &Camp_ope
      Endif
  EndCase
  Skip
Enddo
Return
```

```
Procedure Selec_U
Parameters Archivo, Operando, Condicion, Arch_a

Use &Archivo
Go Top
Select 2
Use &Arch_a
Go Top
Select 1
Do While .Not. Eof()
  Do Case
    Case Operando = "="
      If .Not. (Val(Trim(&Archivo)) = Val(Trim(Condicion)))
        Replace Imprimir With .F.
        Select 2
        Replace Vive With 'F'
        Select 1
      EndIf
    Case Operando = "<"
      If .Not. (Val(Trim(&Archivo)) < Val(Trim(Condicion)))
        Replace Imprimir With .F.
        Select 2
        Replace Vive With 'F'
        Select 1
      EndIf
    Case Operando = ">"
      If .Not. (Val(Trim(&Archivo)) > Val(Trim(Condicion)))
        Replace Imprimir With .F.
        Select 2
        Replace Vive With 'F'
        Select 1
      EndIf
    Case Operando = "<="
      If .Not. (Val(Trim(&Archivo)) <= Val(Trim(Condicion)))
        Replace Imprimir With .F.
        Select 2
        Replace Vive With 'F'
        Select 1
    * EndIf
    Case Operando = ">="
      If .Not. (Val(Trim(&Archivo)) >= Val(Trim(Condicion)))
        Replace Imprimir With .F.
        Select 2
        Replace Vive With 'F'
        Select 1
      EndIf
    Otherwise
      If .Not. (Val(Trim(&Archivo)) # Val(Trim(Condicion)))
        Replace Imprimir With .F.
        Select 2
        Replace Vive With 'F'
        Select 1
      EndIf
  EndCase
```

```
    Select 2
    Skip
    Select 1
    Skip
EndDo
Close Databases
Return
```

```
Procedure Proyecc
Parameters Campos, Archivo

Campos = Trim(Campos)
Archivo = Trim(Archivo)
Camp =
Proy =  n'
Proy_f =  n
Proy_g =  n'
If At( -', Campos) = 0
  Camp = Campos
  Camp1 =
  Camp2 = '
  Camp3 =
  Cuent_camp = 1
  Do While At(   , Camp) # 0
    Pos_blan = At(' ', Camp)
    Do Case
      Case Cuent_camp = 1
        Camp1 = Substr(Camp, 1, Pos_blan - 1)
      Case Cuent_camp = 2
        Camp2 = Substr(Camp, 1, Pos_blan - 1)
      Case Cuent_camp = 3
        Camp3 = Substr(Camp, 1, Pos_blan - 1)
    EndCase
    Cuent_camp = Cuent_camp + 1
    Camp = Substr(Camp, Pos_blan + 1, Len(Camp) - Pos_blan)
  Endbo
  Do Case
    Case Cuent_camp = 1
      Camp1 = Camp
    Case Cuent_camp = 2
      Camp2 = Camp
    Case Cuent_camp = 3
      Camp3 = Camp
  EndCase
  Use &Archivo
  Do Case
    Case Cuent_camp = 1
      Display All &Camp1
    Case Cuent_camp = 2
      Display All &Camp1 , &Camp2
    Case Cuent_camp = 3
      Display All &Camp1 , &Camp2 , &Camp3
  EndCase
  Close Databases
  Archivo = Archivo +  .dbf'
  Erase &Archivo
  Wait
  Return
Endlf
Do While At('-', Campos) # 0
  Pos_guion = At('-', Campos)
  If Pos_guion = Len(Campos) .Or. Substr(Campos, Pos_guion + 1, 1) =
```

```
    Camp = Camp + Substr(Campos, 1, Pos_guion - 1) + ','
    Proy = 's'
    If Len(Campos) - Pos_guion > 3
      Campos = Substr(Campos, Pos_guion + 2,;
                      Len(Campos) - Pos_guion - 1)
    Else
      Exit
    EndIf
  Else
    If Substr(Campos, Pos_guion + 1, 1) = 'A'
      Proy_g = 's'
      If Len(Campos) - Pos_guion > 4
        Campos = Substr(Campos, Pos_guion + 3,;
                        Len(Campos) - Pos_guion - 2)
      Else
        Exit
      EndIf
    Else
      Proy_f = 's'
      Fun = Substr(Campos, Pos_guion + 1, 3)
      If Len(Campos) - Pos_guion > 6
        Campos = Substr(Campos, Pos_guion + 5,;
                        Len(Campos) - Pos_guion - 4)
      Else
        Exit
      EndIf
    EndIf
  EndIf
EndDo
Do Case
  Case Proy = 's' .And. Proy_f = 'n' .And. Proy_g = 'n'
    Camp = Substr(Camp, 2, Len(Camp) - 2)
    Camp1 = ''
    Camp2 = ''
    Camp3 = ''
    Camp4 = ''
    Cuent_camp = 1
    Do While At( ',', Camp) # 0
      Pos_coma = At( ',', Camp)
      Do Case
        Case Cuent_camp = 1
          Camp1 = Substr(Camp, 1, Pos_coma - 1)
        Case Cuent_camp = 2
          Camp2 = Substr(Camp, 1, Pos_coma - 1)
        Case Cuent_camp = 3
          Camp3 = Substr(Camp, 1, Pos_coma - 1)
        Otherwise
          Camp4 = Substr(Camp, 1, Pos_coma - 1)
      EndCase
      Cuent_camp = Cuent_camp + 1
      Camp = Substr(Camp, Pos_coma + 1, Len(Camp) - Pos_coma)
    EndDo
    Do Case
      Case Cuent_camp = 1
```

```
      Camp1 = Camp
    Case Cuent_camp = 2
      Camp2 = Camp
    Case Cuent_camp = 3
      Camp3 = Camp
    Otherwise
      Camp4 = Camp
  EndCase
  Use &Archivo
  If Lastrec() < 1
    @1,1 Say ' Ningun registro recuperado
  EndIf
  Do Case
    Case Cuent_camp = 1
      Display All &Camp1
    Case Cuent_camp = 2
      Display All &Camp1 , &Camp2
    Case Cuent_camp = 3
      Display All &Camp1 , &Camp2 , &Camp3
    Otherwise
      Display All &Camp1, &Camp2, &Camp3, &Camp4
  EndCase
  Close Databases
  Archivo = Archivo +  .dbf
  Erase &Archivo
Case Proy =  n  .And. Proy_f =  s  .And. Proy_g =  n
  Do Case
    Case Fun = 'SUM'
      Arch_f =  Suma
    Case Fun =  CNT
      Arch_f =  Cuenta
    Case Fun =  PRM'
      Arch_f = 'Promedio'
    Case Fun =  MAX
      Arch_f = 'Maximo'
    Case Fun =  MIN
      Arch_f = 'Minimo
  EndCase
  Use &Arch_f
  If Lastrec() < 1
    @1,1 Say ' Ningun registro recuperado'
  EndIf
  Go top
  Display All &Arch_f For Imprimir
  Close Databases
  Arch_f = Arch_f +  .dbf'
  Erase &Arch_f
Case Proy =  n  .And. Proy_f =  n  .And. Proy_g =  s
  Use &Archivo
  Copy To tempo For Vive =  V
  Select 2
  Use tempo
  If Lastrec() < 1
    @1,1 Say ' Ningun registro recuperado
```

```
  EndIf
  Select 1
  Display All Cam_agru, Contenid For Vive = 'V
  Close Databases
  Archivo = Archivo + '.dbf'
  Erase &Archivo
  Erase tempo.dbf
Case Proy_f = 's' .And. Proy_g = 's'
  Do Case
    Case Fun = 'SUM'
      Arch_f = 'Suma'
    Case Fun = 'CNT'
      Arch_f = 'Cuenta'
    Case Fun = 'PRM'
      Arch_f = 'Promedio'
    Case Fun = 'MAX'
      Arch_f = 'Maximo'
    Otherwise
      Arch_f = 'Minimo'
  EndCase
  Use &Archivo
  Copy To Estruc Structure Extended
  Select 2
  Use Estruc
  Go Bottom
  Append Blank
  Replace Field_Name With 'Resultad', Field_Type With 'Character',;
          Field_Len With 10
  Close Databases
  Create Listado From Estruc
  Append From &Archivo
  Select 1
  Use Listado
  Select 2
  Use &Arch_f
  Select 1
  Do While .Not. Eof()
    Replace Resultad With B -> &Arch_f
    Skip
    Select 2
    Skip
    Select 1
  EndDo
  Copy To tempo For Vive = 'V'
  Select 2
  Use tempo
  If Lastrec() < 1
    @1,1 Say 'Ningun registro recuperado'
  EndIf
  Select 1
  Display All Cam_agru, Contenid, Resultad For Vive = 'V'
  Arch_f = Arch_f + '.dbf'
  Archivo = Archivo + '.dbf'
  Close Databases
```

```
        Erase Estruc.dbf
        Erase Listado.dbf
        Erase tempo.dbt
        Erase &Arch_f
        Erase &Archivo
EndCase
Wait
Return
```

```
Procedure Case2
Parameters Rel_1, Rel_2, Rel_3, Rel_4, Cade_1, Cade_2, Cade_3, Cade_4

Cade_1 = '0
Cade_2 =  0'
Cade_3 = '0'
Cade_4 =  0'
Contador = 1
Do While Contador <= 4
  Do Case
     Case Contador = 1
          Relacion = Rel_1
     Case Contador = 2
          Relacion = Rel_2
     Case Contador = 3
          Relacion = Rel_3
     Case Contador = 4
          Relacion = Rel_4
  EndCase
  If Substr(Relacion, 1, 1) =
    Exit
  Else
    Do Case
       Case Substr(Relacion, 1, 2) =  CC
            Cadena = '2'
       Case Substr(Relacion, 1, 2) = 'TR'
            Cadena =  3
       Otherwise
            Cadena =
            Do Gen_Rela With Relacion, Cadena
    EndCase
  EndIf
  Do Case
      Case Contador = 1
           Cade_1 = Cadena
      Case Contador = 2
           Cade_2 = Cadena
      Case Contador = 3
           Cade_3 = Cadena
      Case Contador = 4
           Cade_4 = Cadena
  EndCase
Contador = Contador + 1
EndDo
Return

* ------------------------------------------------------------------
* ------------------------------------------------------------------

Procedure Gen_Rela
Parameters Relacion, Cadena

Relacion = Trim(Relacion)
Cadena = Cadena + Relacion +
```

```
Use &Relacion
Copy to Estruct Structure Extended
Select 2
Use Estruct
Go Bottom
Num_rec = Recno()
Go 1
i = 1
Cont_sep = 1
Do While i <= Num_rec
   Cadena = Trim(Cadena+Field_Name)+ |
   Cont_sep = Cont_sep + 1
   i = i + 1
   Skip
EndDo
Do Case
   Case Cont_sep = 2
        Cadena = Cadena + |!&'
   Case Cont_sep = 3
        Cadena = Cadena +  !&'
   Case Cont_sep = 4
        Cadena = Cadena +  &'
   Case Cont_sep = 5
        Cadena = Substr(Cadena, 1, Len(Cadena) - 1) +  &
EndCase
Cadena = Substr(Cadena, 2, Len(Cadena))
Close Databases
Erase Estruct.dbf
Return
```

```
      ****    PROCEDIMIENTOS EN C DEL ARCHIVO QBE_C.ISI     *****

/* Este Programa generara un ARBOL DE PARSE para una solicitud en     */
/*            *********  QUERY By EXAMPLE  **********                 */
/*                    y realiza su recorrido                          */

/* Este es el Archivo principal del Programa                          */
/* En el esta definido:                                               */
/*          PROGRAMA PRINCIPAL                                      */
/* y las siguientes funciones:                                        */


/* Declaracion de las MACRO DEFINICIONES                              */

#define Max_tablas (10)    /* define numero maximo de tablas          */
                           /* en una solicitud QBE                    */
#define T_L (80)           /* define Tamano Max. una Linea de Solicitud  */

#define Nom_rela (16)      /* define Tamano Max. de nom. de Relacion     */

#define Tam_entrada (70)   /* Tamano maximo de entradas de una linea     */

/* definicion de NULL (apunatdor a  nada ) en  stdio.h                */

/* Declaraciond de variables EXTERNAS                                 */

struct Hoja_arbol
  {
  struct Nodo_arbol *Apun_padre; /* Apunta a un nodo no Hoja */
  char Nom_tabla[Nom_rela];      /* Tiene nombre  tabla corresp */
  char Entradas[Tam_entrada];    /* Entradas en la linea */
  char Tabla_unib;               /* Bandera indica si tabla es unible */
  char Caja_cond;                /* Band. indica si linea hay C. condi. */
  char Tabla_resul;              /* Band. indica si linea es Tabla Resul. */
  }  Nodo_hoja[Max_tablas];
 * Tipo de tipo Hoja del arbol */

struct Entrada_QBE
  {
  char Solicitud[T_L]; /* linea completa de Solicitud en QBE */
  struct Nodo_arbol *Apun_ultimo;
  /* Apunta ultimo nodo desarrolado por algoritmo apartir */
  /* de su nodo hoja correspondiente */
  int Ap_i_e_l;  /* Apun. primer caract. extre. izq. de entradas */
  int Ap_d_e_l;  /* Apun. ultimo caract. estre. der. de entradas */
  int Ap_p_c_e;  /* Apun. primer caract. de una entrada */
  int Ap_u_c_e;  /* Apun. ultimo caract. de una entrada */
  } Entradas[Max_tablas];
  /*Registro que tendra un tabla (linea) de la solicitud */

struct Nodo_arbol
  {
  struct Nodo_arbol *Apunta_padre; /* Apunta al nodo padre */
  char Oper_realizar[3];           /* Caracteres de op. a realizar  */
```

```
  char Operandos[50];              /* Operandos de la operacion */
  char In_one_file;
  /* Indica si datos de entrada vienen de un solo archivo     */
  char Out_one_file;
  /* Indica si la salida se dejara en un solo Archivo         */
  struct Nodo_arbol *Hijo_der, *Hijo_izq;
  /* Apunta a los hijos izq. y der. del nodo                  */
  int Apun_pad, Apun_hoi;
  /* Apuntador a nodo hoja hijo derecho e izquierdo           */
  char Arch_sal[10];
  /* Archivo salida que contendra resultado de la operacion */
  int Num_ar_s;  /* Num. de Archivo de salida                 */
  ) ;
     /* Apuntador para variables de tipo Nodo_arbol (Nodos)        */

     int Num_line;  /* Define numero lineas que forman solicitud  */

/* Definicion de variables globales del Recorrido del Arbol          */

char Ramas_reco[Max_tablas] = {'n',  n ,  n',  n ,  n ,  n ,  n ,  n ,
                               n ,  n }; /* Indica si rama asociada fue */
                                         /* recorrida */
struct Nodo_arbol *Nodo_anali[3]; /* Apunta nodo en analisis. Solo 3 sub */
                                  /* arboles */
int Rama_reco = 0;     /* Primer rama a recorrer del arbol */

int Num_sub = 0;       /* Numero de sub-arboles analizados */

char Pendiente=  n ;  /* Indica algun nodo dos hijos pendiente de resolver */

char Oper_pendi[3];    /* Operacion pendiente a analizar */

char Operan_pendi[50];  /* Operandos pendiente de completar */

char Archivo_pendi[10]; /* Uno dos archivo sobre realizar oper. pendiente */

struct Nodo_arbol *Nodo_pendi;  /* Apunta Nodo con oper. pendiente analiz. */

char Lineas[4][f_L];   /* Contiene a la solicitud en forma Lineal */

#include "stdio.h"

char *malloc(), *strcpy(), *strcat();
```

```
/*****************************************************/
/*                                                   */
/*            PROGRAMA ARBOL                         */
/*                                                   */
/*****************************************************/

Arbol()
/*  Par_no_gr  Fun. defi. ext. y parsea fun. no implican Grupos */
  {
        Get_tablas();
        Sep_tabla();
        Veli_entr();
        Busca_CC();
        Busca_IK();
        Par_no_gr();
        Par_Grup();
  } /*Fin Arbol*/
/**/

/* Funcion que inicializa las variables globales que se tiene */
/*****************************************************/
/*                                                   */
/*            SUBRUTINA  INI_VA_G                    */
/*                                                   */
/*****************************************************/

Ini_va_g()
  {
  int i, j, lon;
  char *ap;

  ap = "                                                  ";
  lon = Nom_rela;
  for (i = 0; i <= (Max_tablas - 1); i++) /* Ini. var. Nodo_Hoja */
    {
    strcpy(Nodo_hoja[i].Nom_tabla, ap, lon);
    Nodo_hoja[i].Tabla_unib =  s ;
    Nodo_hoja[i].Apun_padre = (struct Nodo_arbol *) NULL;
    Nodo_hoja[i].Caso_cond =  n ;
    Nodo_hoja[i].Tabla_resul =  n ;
    Nodo_hoja[i].Nom_tabla[Nom_rela - 1] = '\0' ;
    for (j = 0; j <= 48; Nodo_hoja[i].Entradas[j] =    , j++);
    Nodo_hoja[i].Entradas[49] = '\0' ;
    }  /* Inicializa campos de elementos de vec. Nodo_hoja */
  lon = I_L;
  for (i = 0; i <= (Max_tablas - 1); i++) /* Ini. var. Entradas */
    {
    strcpy(Entradas[i].Solicitud, ap, lon);
    Entradas[i].Apun_ultimo = (struct Nodo_arbol *) NULL;
    Entradas[i].Ap_i_e_l = Entradas[i].Ap_d_e_l =
    Entradas[i].Ap_p_c_e = Entradas[i].Ap_u_c_e = 0;
    }  /* Inicializa campos de elementos de vec. Entradas */
  } /*Fin Ini_va_g*/
/**/
```

```
/* Funcion que lee las tablas de la solicitud del usuario *.
/*****************************************************************/
/*                                                               */
/*            SUBRUTINA  GET_TABLAS                              */
/*                                                               */
/*****************************************************************/

Get_tablas()
  {
  char *men_1;
  int i, j;

  Ini_va_g();
  for (i=0; i <= Num_line; i++)
    strcpy(Entradas[i].Solicitud, Lineas[i]);
  } /*Fin Get_tablas*/
/**/


/* Separa el nombre de la tabla (Relacion) de cada linea de Solicitud *
/* y las Entradas que aparecen en la misma                           */
/*****************************************************************/
/*                                                               */
/*            SUBRUTINA  SEP_TABLA                               */
/*                                                               */
/*****************************************************************/

Sep_tabla()
  {
  int i, j, k;

  i = 0;
  do
    {
    for
      (
      k= 0, j = (Entradas[i].Solicitud[0] ==  ' ' )?
             ( Nodo_hoja[i].Tabla_unib =  'n' , 2): 0;
             Entradas[i].Solicitud[j] !=  '(' ; j++, k++
      )
      Nodo_hoja[i].Nom_tabla[k] = Entradas[i].Solicitud[j];
    Nodo_hoja[i].Nom_tabla[k] =  '\0' ;
    for (k = 0 ; Entradas[i].Solicitud[j] !=  ')' ; j++, k++)
      Nodo_hoja[i].Entradas[k] = Entradas[i].Solicitud[j];
    Nodo_hoja[i].Entradas[k] = Entradas[i].Solicitud[j];
    Nodo_hoja[i].Entradas[++k] =  '\0' ;
    } while (Entradas[++i].Solicitud[2] !=  '\0' );
  }    /*Fin Sep_tabla*/
/**/
```

```
* Coloca apuntadores en c/linea delimitando los extremos de la subca-
  cadena de la linea en la que estan las entradas de la misma */
/*****************************************************************/
/*                                                               */
/*            SUBRUTINA  DELI_ENTR                               */
/*                                                               */
/*****************************************************************/

eli_entr()
 {
 int i, j;

 j = 0;
 do
   {
   for (j = 0; Entradas[i].Solicitud[j] !=  ( ; j++);
   Entradas[i].Ap_i_e_l = ++j;
   for ( ;Entradas[i].Solicitud[j] !=  ) ; j++);
   Entradas[i].Ap_d_e_l = --j;
   } while (Entradas[++i].Solicitud[1] !=    );
 } /*Fin Dele_entr*/
**/

* Busca por la linea que corresponda a la de una Caja de Condicion */
/*****************************************************************/
/*                                                               */
/*            SUBRUTINA  BUSCA_CC                                */
/*                                                               */
/*****************************************************************/

usca_CC()
 {
 int i;
 char *Cadena;

 Cadena = "Caja Condicion ";
 for (i = 0; i <= Num_line; i++)
   if ((strcmp(Nodo_hoja[i].Nom_tabla, Cadena)) == 0 )
     Nodo_hoja[i].Caja_cond =  s ;       /* Fin de For */
 } /*Fin Busca_CC*/
**/

* Busca por la linea que corresponda a la de una Tabla  */
* de Resultados                                         */
/*****************************************************************/
/*                                                               */
/*            SUBRUTINA  BUSCA_TR                                */
/*                                                               */
/*****************************************************************/

Busca_TR()
 {
 int i;
 char *Cadena;
```

```
  Cadena = "Tabla Resulta  ";
  for (i = 0; i <= Num_line; i++)
    if ((strcmp(Nodo_hoja[i].Nom_tabla, Cadena)) == 0 )
      Nodo_hoja[i].tabla_resul =  z ;       /* Fin de For */
  } /*Fin Busca_IR*/
/**/
```

```
/* Localizara una entrada en alguna solicitud que contiene una subcadena */
/* de interes. La entrada encontrada la copiara a otra cadena */
/*****************************************************************/
/*                                                               */
/*            SUBRUTINA  BUSCA_CA                                */
/*                                                               */
/*****************************************************************/

char Busca_ca(Cadena, Sub_cad, Sub_busca)

char *Cadena;    /* Cadena en la que se esta buscando */
char *Sub_cad; /* Sub-cadena tendra la entrada con sub buscada */
char *Sub_busca;  /* Sub-cadena por la que se busca */

  {
  int Indice_L;
  /* Apuntadores a cadena de linea y a la sub cadena buscada */
  char Banderita;  /* Indica si la cadena fue encontrada */
  char Compara();

  Banderita = Compara(Cadena, Sub_busca, &Indice_L);
  if (banderita ==  n )
    return(Banderita);
  Sep_ent(Cadena, Sub_cad, Indice_L);
  return(Banderita);
  }    /*Fin Busca_ca*/
/**/

/* Separara de una linea, una entrada que tiene una determinada sub- */
/* cadena. Tambien Elimina de la Cadena la parte ya explorada es decir */
/* la actualiza.                                                      */
/*****************************************************************/
/*                                                               */
/*            SUBRUTINA  SEP_ENT                                 */
/*                                                               */
/*****************************************************************/

Sep_ent(Cadena, Sub_cad, Indice_L)

char *Cadena, *Sub_cad;
int Indice_L;

  {
  int apuntito;
  int ap_atras, ap_adelante;

  ap_atras = ap_adelante = Indice_L;
  for (--ap_atras; Cadena[ap_atras] !=  ,  && Cadena[ap_atras]
       !=  ( ; ap_atras--);
  ++ap_atras;
  for (++ap_adelante; Cadena[ap_adelante] !=  ,  &&
       Cadena[ap_adelante] != ') ; ap_adelante++);
  --ap_adelante;
  for (apuntito = 0, Indice_L = ap_atras; Indice_L <= ap_adelante;
```

```
      apuntito++, Indice_L++)
    Sub_cad[apuntito] = Cadena[Indice_L];
  Sub_cad[apuntito] =  \0 ;
  if (Cadena[Indice_L] ==  ) )
    Cadena[0] =  \0 ;
  else
    {
    char Cad_temp[Tam_entrada];

    strcpy(Cad_temp, Cadena);
    for(apuntito = 0; apuntito <= 49; apuntito++)
      Cadena[apuntito] =    ;
    for(apuntito = 0; Cad_temp[Indice_L] !=  ) ;
        apuntito++, Indice_L++)
      Cadena[apuntito] = Cad_temp[Indice_L];
    Cadena[apuntito++] = Cad_temp[Indice_L];
    Cadena[apuntito] =  \0 ;
    }                          /* Fin del  else */
  }     /*Fin del Sep_ent*/
/**/

/* Creara en una forma Dinamica un Nodo Hoja del Arbol */
/*****************************************************/
/*                                                   */
/*            SUBRUTINA CREA_NODO                    */
/*                                                   */
/*****************************************************/

struct Nodo_arbol *Crea_nodo()

  {
  int i;
  struct Nodo_arbol *Ap_nodo;

  Ap_nodo= (struct Nodo_arbol *) malloc(sizeof(struct Nodo_arbol));
  Ap_nodo->Apunta_padre = (struct Nodo_arbol *) NULL;
  for (i=0; i <= 1;  i++) Ap_nodo->Oper_realizar[i] =  ' ;
  Ap_nodo->Oper_realizar[2] =  \0 ;
  for (i =0; i <= 49; i++) Ap_nodo->Operandos[i] =    ;
  Ap_nodo->Operandos[0] = '\0 ;
  Ap_nodo->Out_one_file =    ;
  Ap_nodo->In_one_file =    ;
  Ap_nodo->Hijo_der = Ap_nodo->Hijo_izq = (struct Nodo_arbol *)NULL;
  Ap_nodo->Apun_hoi = Ap_nodo->Apun_hod = 999;  /* Ap. indice no existe */
  for(i = 0 ; i <= 8; Ap_nodo->Arch_sal[i] =    , i++);
  Ap_nodo->Arch_sal[0] = '\0 ;
  Ap_nodo->Num_ar_s = 0;  /* No tiene algun archivo de salida */
  return(Ap_nodo);
  }    /*Fin Crea_nodo*/
/**/
```

```
   De una cadena de caracteres de una entrada extraera el nombre de la */
/* Relacion (Archivo) y el elemento que aparecen en ella */
/*****************************************************************/
/*                                                               */
/*             SUBRUTINA SEP_INFOR                               */
/*                                                               */
/*****************************************************************/

Sep_infor(Sub_cad, Camp_op, Elemento, Relacion)

char *Sub_cad;
char *Camp_op;
char *Elemento;
char *Relacion;

  {
  int i, j, k;
  char Bus_op_rel();

  for (i = 0; Sub_cad[i] != '\0' ; i++);
  j = --i;
  for ( ;Sub_cad[i] != '.' && Sub_cad[i] != ':' ; i--);
  *Relacion = Bus_op_rel (Sub_cad, i);
  i = (*Relacion == '=' ) ? i + 2: i + 4;
  for (k = 0; i <= j; k++, i++)
    Elemento[k] = Sub_cad[i];  /* Se encontro elemen. de entrada */
  Elemento[k] = '\0' ;
  for (i = (Sub_cad[0] == ' ') ? 1: 0, j = 0;
            Sub_cad[i] != ':' ; i++, j++)
    Camp_op[j] = Sub_cad[i];
  /* Se separa el campo de la Entrada */
  Camp_op[j] = '\0' ;
  }    /*Fin Sep_infor*/
/**/

/* Coloca los operandos en el campo  Operandos  del nodo hoja  corres- */
/* pondiente                                                          */
/*********************************************************************/
/*                                                                   */
/*             SUBRUTINA FUN_OPERAN                                  */
/*                                                                   */
/*********************************************************************/

fun_operan(Ap_nodo, Camp_op, Relacion, Elemento, Apn_opera, Op_lo)

struct Nodo_arbol *Ap_nodo;
char *Camp_op;
char Relacion;
char *Elemento;
int *Apn_opera;  /* Apunta al siguiente caract. a escribir */
char Op_lo;

 {
 int i;
```

```
if (*Apn_opera != 0)
  {
  Ap_nodo->Operandos[(*Apn_opera)++] = Op_lo;
  Ap_nodo->Operandos[*Apn_opera] = Op_lo;
  *Apn_opera += 2;
  }     /*bin if*/
for (i = 0; Camp_op[i] != '\0' ; (*Apn_opera)++, i++)
Ap_nodo->Operandos[*Apn_opera] = Camp_op[i];
++(*Apn_opera);
Ap_nodo->Operandos[(*Apn_opera)++] = Relacion;
for (++*Apn_opera, i = 0; Elemento[i] != '\0' ; (*Apn_opera)++, i++)
Ap_nodo->Operandos[*Apn_opera] = Elemento[i];
(*Apn_opera)++;
Ap_nodo->Operandos[*Apn_opera] = '\0' ;
}/*Pon_operan*/
*/
```

```
/*esta subrutina compara una cadena con su patron*/
/*retorna un 0 si son iguales y un 1 si no*/
/*************************************************************/
/*                                                           */
/*            SUBRUTINA  ESIGUAL                             */
/*                                                           */
/*************************************************************/

esigual(Cadena,Patron)

char *Cadena, *Patron;

 {
 int Valor, i;

 Valor = 0;
 for (i = 0; Patron[i] != '\0' ; i++)
   {
   if (Cadena[i] != Patron[i] )  {Valor = 1; break;}
   }
 return (Valor);
 }    /*Fin esigual*/
**/

/*esta subrutina compara "cadena" con un "patron"*/
/*si lo pudo encontrar, hace que *respuesta sea 's' */
/* y hace que en "pos" aparezca la posicion en que*/
/*comienza "patron" dentro de cadena*/
/* si no lo encuentra, hace que *respuesta sea 'n' */
/* y que *pos sea (-1)*/
/*************************************************************/
/*                                                           */
/*            SUBRUTINA  COMPARA                             */
/*                                                           */
/*************************************************************/

char Compara(Cadena, Patron, Pos)

char *Cadena,*Patron;
int *Pos;

 {
 char Respuesta;
 int Halle = 999;
 int Lim, k;

 Lim = strlen(Cadena) - strlen(Patron);
 for (k = 0; k<=Lim && (Halle = Esigual(&Cadena[k], Patron)) != 0; k++);
 if (Halle == 0) {Respuesta = 's' ; *Pos = k;}
 else {Respuesta = 'n' ;*Pos = -1;}
 return(Respuesta);
 }    /*Fin compara*/
**/
```

```
/*Esta subrutina compara una cadena con su patron*/
/*retorna un 0 si son iguales y un 1 si no*/
/*****************************************************/
/*                                                   */
/*            SUBRUTINA  ESIGUAL                     */
/*                                                   */
/*****************************************************/

esigual(Cadena,Patron)

char *Cadena, *Patron;

  {
  int Valor, i;

  Valor = 0;
  for (i = 0; Patron[i] != '\0' ; i++)
    {
    if (Cadena[i] != Patron[i] )  {Valor = 1; break;}
    }
  return (Valor);
  }    /*Fin esigual*/
/**/

/*esta subrutina compara "cadena" con un "patron"*/
/*si lo pudo encontrar, hace que *respuesta sea  s'*/
/* y hace que en "pos" aparezca la posicion en que*/
/*comienza "patron" dentro de cadena*/
/* si no lo encuentra, hace que *respuesta sea  n */
/* y que *pos sea (-1)*/
/*****************************************************/
/*                                                   */
/*            SUBRUTINA  COMPARA                     */
/*                                                   */
/*****************************************************/

char Compara(Cadena, Patron, Pos)

char *Cadena,*Patron;
int *Pos;

  {
  char Respuesta;
  int Halle = 999;
  int Lim, k;

  Lim = strlen(Cadena) - strlen(Patron);
  for (k = 0; k<=Lim && (Halle = Esigual(&Cadena[k], Patron)) != 0; k++);
  if (Halle == 0) {Respuesta =  's' ; *Pos = k;}
  else {Respuesta =  'n' ;*Pos = -1;}
  return(Respuesta);
  }    /*Fin compara*/
/**/
```

```
* Funcion que encontrara, en una entrada de una linea, *
* Operador Relacional                                   *
*********************************************************
*                                                       *
*            SUBRUTINA BUS_OP_REL                       *
*                                                       *
*********************************************************/

har Bus_op_rel(Sub_cad, i)

define car Sub_cad

har *Sub_cad;
nt i;

 {
 if (car[i + 2] ==  >  || car[i + 2] ==  <  || car[i + 2] ==  =  )
   return(car[i + 2]);
 else
   return( = );
 }    /*Fin Bus_op_rel*/
ndef car
**/

* Encadena el nodo recien creado con sus hijos, ya sean */
* otros nodos o hojas. Si  hijo = 2  los encadena al hi-*/
* jo derecho, y si el igual a  i  los encadena al hijo  */
* izquierdo.                                            */
*********************************************************
*                                                       */
*            SUBRUTINA ENCA_NODO                        */
*                                                       */
*********************************************************/

nca_nodo(Ap, i, hijo)

truct Nodo_arbol *Ap;
nt i, hijo;

 {
 struct Nodo_arbol *Ap_aux;

 Entradas[i].Apun_ultimo = Ap;
 if (Nodo_hoja[i].Apun_padre == (struct Nodo_arbol *) NULL))
   {
   Nodo_hoja[i].Apun_padre = Ap;
   if (hijo == 2)
     Ap->Apun_hod = i;
   else
     Ap->Apun_hoi = i;
   }
 else
   {
   Ap_aux = Nodo_hoja[i].Apun_padre;
```

```
    while (Ap_aux->Apunta_padre != ((struct Nodo_arbol *) NULL))
      Ap_aux = Ap_aux->Apunta_padre;
    if (hijo == 2)
      Ap->Hijo_der = Ap_aux;
    else
      Ap->Hijo_izq = Ap_aux;
    Ap_aux->Apunta_padre = Ap;
    } /*Fin i*/
  }  /*Fin Enca_nodo*/
/**/

/* Determinara si un Elemento ejemplo encontrado en una  */
/* Linea de Solicitud, ya fue analizado                  */
/*********************************************************/
/*                                                       */
/*             SUBRUTINA ANA_ANTES                       */
/*                                                       */
/*********************************************************/

char Ana_antes(i, Elemento)

int i;
char *Elemento;

  {
  int k;
  char Bandera =  n ;     /* Bandera en que se In/Out una respuesta */
  int Posi;
  char Compara();

  for (k = 0; k <= i; k++)
    {
    Bandera = Compara(Nodo_hoja[k].Entradas, Elemento, &Posi);
    if (Bandera == 's') break;
    }  /*Fin for*/
  return(Bandera);
 }  /*Fin Ana_antes*/
/**/
```

```
            *Num_fun = 2;
          else if ((Bandera = Compara(Cadena, "MAX.", &Posi)) ==  s )
                 *Num_fun = 3;
               else
                 {
                 Bandera = Compara(Cadena, "MIN.", &Posi);
                 if (Bandera ==  s )
                   *Num_fun = 4;
                 }
 return(Bandera);
 }  /*Fin Hay_fun*/
*/
```

```
/* Esta Seccion contiene solo funciones que se encargaran de Parsear *
 * Palabras Claves que implican funciones de Grupo.                  */


/* Funcion Maestra que llamara a otras para que en total parsean las *
 * operaciones que implican grupos                                   */
/*********************************************************************/
/*                                                                   */
/*            SUBRUTINA PAR_GRUP                                     */
/*                                                                   */
/*********************************************************************/

Par_Grup()

 {
 int i, j;
 int Line_FF;
 int Num_fun;
 char Ban_fun_CC =  n ;
 char Ban_fun =  n ;
 char Ban_no_un =  n ;
 char Proy_uni();
 char Hay_fun();

 for (j = 0; j <= Num_line; j++)
   if (Nodo_hoja[j].Tabla_unib ==  n )
     {
     Ban_no_un =  s ;
     break;
     } /*Fin if Nodo_hoja*/
 if (Ban_no_un ==  n )
   {
   Agrupa();
   Join_grupos();
   Ban_fun_CC = Proy_uni();
   for (i = 0; i <= Num_line; i++)
     {
     Ban_fun = Hay_fun(Nodo_hoja[i].Entradas, &Num_fun);
     if (Ban_fun ==  s ) break;
     } /*Fin for*/
   if (Ban_fun ==  s')
     Line_FF = Fun_pre();
   if (Ban_fun_CC == 's')
     Sel_grupi(Line_FF);
   } /*Fin if Ban_no_un*/
 else
   {
   Rela_equi();
   Ban_fun_CC = Proy_uni();
   } /*Fin else Ban_no_un*/
 Proy_final();
 }/*Fin Par_Grup*/
/**/
```

```
/* Localiza todos las operaciones de agrupamiento que se */
* encuentren en la solicitud, ya sea que se agrupen por */
/*  Nada  o por algun campo en especial.                  */
/*********************************************************/
/*                                                       */
/*            SUBRUTINA AGRUPA                           */
/*                                                       */
/*********************************************************/

agrupa()

 {
 int i, l;
 char Cadena[Tam_entrada];  /* Entradas de una linea a analizar */
 char Sub_busca[6];         /* Cadena indica si la relacion se agrupara */
 char Sub_cad[30];          /* Contiene la entrada a analizar */
 char Camp_op[10];          /* Campo por el que se agrupara */
 char Elemento[15];
 char Relacion =  = ';
 char Banderita, Band_A;
 int Posi;
 struct Nodo_arbol *Ap_nodo;
 char Busca_ca();
 char Compara();
 struct Nodo_arbol *Crea_nodo();

 for(i = 0; i <= Num_line; i++)
   {
   if (Nodo_hoja[i].Tabla_resul ==  s ) continue; /* Es Tabla Resul */
   if (Nodo_hoja[i].Caja_cond ==  s ) continue;  /* Es Caja Condicion */
   strcpy(Cadena, Nodo_hoja[i].Entradas);
   strcpy(Sub_busca, "todo.");
   Banderita = Compara(Cadena, Sub_busca, &Posi);
   Band_A = Compara(Cadena, "A.", &Posi);
   if (Banderita ==  n  && Band_A ==  n ) continue;
   Ap_nodo = Crea_nodo();   /* Crea Nodo */
   Ap_nodo->Oper_realizar[0] =  A ;
   Enca_nodo(Ap_nodo, i, 2);
   strcpy(Sub_busca, "A.");
   Banderita = Busca_ca(Cadena, Sub_cad, Sub_busca);
   if (Banderita ==  n )
     {
     Ap_nodo->Operandos[0] = 'N ; Ap_nodo->Operandos[1] = 'a ;
     Ap_nodo->Operandos[2] =  d'; Ap_nodo->Operandos[3] =  a ;
     Ap_nodo->Operandos[4] =  \0 ;
     continue;
     }  /*Fin if Banderita*/
   Sep_infor(Sub_cad, Camp_op, Elemento, &Relacion);
   for(l = 0; (Ap_nodo->Operandos[l] = Camp_op[l]) !=  \0';
       l++);  /* Pone operandos */
   } /*Fin for i*/
 }  /*Fin Agrupa*/
/**/
```

```
/* Localizara todas las operaciones de  Join  de grupos  */
/* que aparescan en la solicitud, realizando las opera-  */
/* ciones correspondientes. Creara los nodos del arbol   */
/* correspondientes.                                     */
/*********************************************************/
/*                                                       */
/*            SUBRUTINA JOIN_GRUPOS                      */
/*                                                       */
/*********************************************************/

join_grupos()

 {
 int i;
 char Cadena[Num_entrada];  /* Entradas a analizar de una linea */
 char Sub_busca[6];         /* Cont. cadena que indica la entr. analizar */
 char Sub_cad[30];          /* Contiene la entrada a analizar */
 char Banderita;
 struct Nodo_arbol *Crea_nodo();
 char Compara();
 char Busca_ca();
 char Ana_antes();

 for (i = 0; i <= Num_line; i++)
   {
   if (Nodo_hoja[i].Tabla_resul ==  's' ) continue; /* Si TR no analiza */
   if (Nodo_hoja[i].Caja_cond ==  's' )  continue; /* Si CC no analiza */
   strcpy(Cadena, Nodo_hoja[i].Entradas);
   strcpy(Sub_busca, "todo.");
   Banderita = Busca_ca(Cadena, Sub_cad, Sub_busca);
   if (Banderita ==  's')
     {
     char Relacion = '= ;
     char Camp_op[10];  /* Campo al que se le hara la operacion */
     char Elemento[20];   /* Elemento ejemplo de la operacion     */
     char Par_cual =  'n';  /* Indican con  s  si en la entrada   */
     int Posi;
     int Line_par;
   * char Band;

     Sep_infor(Sub_cad, Camp_op, Elemento, &Relacion);
     Band = Compara(Sub_cad, "(", &Posi);
     if (Band ==  's' )
       Par_cual = 's';
     Band = Ana_antes(i - 1, Elemento);
     if (Band ==  's') continue;
     Line_par = Sig_ejem(Elemento, i + 1);
     if (Line_par == -1 || Nodo_hoja[Line_par].Tabla_resul ==  's')
       continue;
     if (Line_par != -1 && Nodo_hoja[Line_par].Caja_cond ==  'n' )
       {
       int Ap_opera = 0; /* Campo Operandos apunta ult. carc. escr. */
       struct Nodo_arbol *Ap_nodo;
       char Par_cua2 =  'n' ;  /* aparecen Parentesis Cuadrados      */
```

```
        char Cadena2[Tam_entrada];
        char Camp_op2[10];
        char Op_logico =  & ;
        char Contenido;

        Ap_nodo = Crea_nodo();
        Ap_nodo->Oper_realizar[0] =  J ; Ap_nodo->Oper_realizar[1] =  q ;
        Ap_nodo->Oper_realizar[2] = '\0' ;
        Enca_nodo(Ap_nodo, i, 2);
        Enca_nodo(Ap_nodo, Line_par, 1);
        strcpy(Cadena2, Nodo_hoja[Line_par].Entradas);
        Banderita = Busca_ca(Cadena2, Sub_cad, Elemento);
        Sep_inter(Sub_cad, Camp_op2, Elemento, &Relacion);
        Band = Compara(Sub_cad, "[", &Posi);
        if (Band ==  s )
          Par_cua2 =  s ;
        if (Par_cual ==  s )  Relacion =  < ;
        else if (Par_cua2 ==  s')  Relacion = '> ;
             else Relacion =  = ;
        Pon_operan(Ap_nodo, Camp_op2, Relacion, Camp_op, &Ap_opera,
                   Op_logico);
        strcpy(Cadena, Nodo_hoja[i].Entradas);
        Band = Compara(Cadena, "].", &Posi);
        if (Band ==  s )      /* La relacion que tenga la Operacion  l. */
Contenido =  d ;    /* sera la que me interese conservar, y    */
        else                  /* significa que la otra me sirve como     */
           Contenido =  i ;   /* patron de comparacion unicamente. La    */
        /*  i  o  d  indica la posicion de la relacion, respecto a la  */
        /* operacion de  Jq , que me interesa conservar.                */
        Ap_nodo->Operandos[Ap_opera++] = Contenido;
        Ap_nodo->Operandos[Ap_opera] =  \0 ;
        )  /*Fin if Line_par*/
      )  /*Fin if Banderita*/
    )  /*Fin for*/
  )  /*Fin Join_grupos*/
**/
```

```
* Realiza una Proyeccion de determinados campos, dando  */
* como resultado una relacion que no tiene tuplas repe- */
* tidas.                                                 */
/*********************************************************/
/*                                                       */
/*            SUBRUTINA PROY_UNI                         */
/*                                                       */
/*********************************************************/

har Proy_uni()

{
int i, j, k;
char Cadena[lam_entrada];  /* Entradas de una linea a analizar */
char Operando[50];      /* Contiene en forma temporal operandos operacion */
char Sub_cad[30];       /* Contiene la entrada a analizar */
char Camp_op[10];       /* Campo al que se le hara la operacion */
char Elemento[15];      /* Elemento ejemplo en la entrada */
char Relacion =  = ;
char Band_i, Band_fun, Bandera;
int Posicion;
char Funcion[5];      /* Funcion encontrada en la Caja de Condicion */
struct Nodo_arbol *Ap_nodo;  /* Apuntador al nodo recien creado */
struct Nodo_arbol *Crea_nodo();
char Compara();
char Busca_ca();
char Hay_fun();

Operando[0] =  \0 ; Band_fun =  n ;
for(i = 0; i <= Num_line; i++)
  {
  if (Nodo_hoja[i].Caja_cond ==  s )
    {
    strcpy(Cadena, Nodo_hoja[i].Entradas);
    Band_fun = Hay_fun(Cadena, &j);
    if (Band_fun ==  n ) continue;
    for (k = i, j = 0; j < 5; k++, j++)
      Funcion[j] = Cadena[k];
  Funcion[j] =  \0 ;
    k = i;  /* Deja apuntador a linea es Caja Condicion */
    continue;
    } /*Fin if Nodo_hoja.Caja_cond*/
  Band_i = Compara(Nodo_hoja[i].Entradas, "1.", &Posicion);
  if (Band_i ==  s )
    {
    strcpy(Cadena, Nodo_hoja[i].Entradas);
    if (Nodo_hoja[i].Tabla_resul != 's )
      {
      while (Cadena[0] !=  \0' && Band_i ==  s )
        {
        Band_i = Busca_ca(Cadena, Sub_cad, "I.");
        Sep_infor(Sub_cad, Camp_op, Elemento, &Relacion);
        strcat(Operando, Camp_op);
        strcat(Operando, " ");
```

```
          Band_j = Compara(Cadena, "1.", &Posicion);
          }  /*Fin while*/
        Loc_fun_pre(i, Operando);
        for(j = 0; j <= Num_line; j++)
          {
          if (j == i || Nodo_hoja[j].Caja_cond ==  's' ) continue;
          Loc_fun_pre(j, Operando);
          } /*Fin for j*/
        Ap_nodo = Creo_nodo();
        Ap_nodo->Oper_realizar[0] =  'P' ; Ap_nodo->Oper_realizar[1] =  'U' ;
        Ap_nodo->Oper_realizar[2] = '\0' ;
        strcpy(Ap_nodo->Operandos, Operando);
        Enca_nodo(Ap_nodo, i, 2);
        } /*Fin if Nodo_hoja.Tabla_resul !=  's' */
      else
        Operando_IR(Cadena, i);
      } /* Fin if Band_j*/
    }  /*Fin for*/
  if (Band_fun ==  's')
    {
    Bandera = Compara(Nodo_hoja[k].Entradas, "E.", &Posicion);
    Posicion += 5;
    for(j = 0; Nodo_hoja[k].Entradas[Posicion] !=  ' ' ; j++, Posicion++)
      Elemento[j] = Nodo_hoja[k].Entradas[Posicion];
    Elemento[j] =  '\0' ;
    for(j = 0; j <= Num_line; j++)
      {
      struct Nodo_arbol *Ap_aux;

      if(j == k ) continue;
      strcpy(Cadena, Nodo_hoja[j].Entradas);
      Bandera = Busca_ca(Cadena, Sub_cad, Elemento);
      if (Bandera ==  's' )
        {
        Sep_infor(Sub_cad, Camp_op, Elemento, &Relacion);
        Ap_aux = Nodo_hoja[j].Apun_padre;
        while (Ap_aux->Apunta_padre != ((struct Nodo_arbol *) NULL))
          Ap_aux = Ap_aux->Apunta_padre;
        Ap_nodo = Ap_aux;
        Band_i = Compara(Ap_nodo->Operandos, Camp_op, &Posicion);
        if (Band_i ==  'n');
          {
          strcpy(Operando, Ap_nodo->Operandos);
          strcat(Operando, Camp_op);
          strcat(Operando, " ");
          strcpy(Ap_nodo->Operandos, Operando);
          } /*Fin if Band_j*/
        break;
        } /*Fin if Bandera*/
      } /*Fin for j*/
    }  /*Fin if Band_fun*/
  return(Band_fun);
  } /*Fin Proy_uni*/
/**/
```

```
* Localiza en una determinada tabla aquellas entradas   */
* que tengan Funciones Prefabricadas.                   */
/*****************************************************/
/*                                                     */
/*            SUBRUTINA LOC_FUN_PRE                    */
/*                                                     */
/*******************************************************/

loc_fun_pre(i, Operandos)

nt i;
har *Operandos;

 {
 int j;
 char Tabla[Tam_entrada];
 char Entrada[30];
 char Camp_op[10];
 char Elemento[15];
 char Relacion =  = ;
 int Posi;
 char Bandera;
 char Compara();
 char Hay_fun();
 char Busca_ca();

 strcpy(Tabla, Nodo_hoja[i].Entrada);
 Bandera = Hay_fun(Tabla, &j);
 while (Tabla[0] !=  \0  && Bandera ==  s )
   {
   switch(j)
     {
     case 0: Bandera = Busca_ca(Tabla, Entrada, "CNT.");
             break;
     case 1: Bandera = Busca_ca(Tabla, Entrada, "SUM.");
             break;
     case 2: Bandera = Busca_ca(Tabla, Entrada, "PRM.");
             break;
     case 3: Bandera = Busca_ca(Tabla, Entrada, "MAX.");
             break;
     case 4: Bandera = Busca_ca(Tabla, Entrada, "MIN.");
             break;
     }  /*Fin switch*/
   Sep_infor(Entrada, Camp_op, Elemento, &Relacion);
   if ((Bandera = Compara(Operandos, Camp_op, &Posi)) ==  n )
     {
     strcat(Operandos, Camp_op);
     strcat(Operandos, " ");
     } /*Fin Bandera =  n */
   Bandera = Hay_fun(Tabla, &j);
   } /*Fin while*/
 } /*Fin Loc_fun_pre*/
*/
```

```
/* Localiza los operandos de la Proyeccion unica cuando  */
/* se tiene una tabla de Resultados.                     */
/*********************************************************/
/*                                                       */
/*            SUBRUTINA OPERANS_IR                       */
/*                                                       */
/*********************************************************/

Operans_IR(Cad, i)

char *Cad;
 nt i;

  {
  int j, k;
  char Tabla[Tam_entrada];  /* Tabla en que se buscara ele. ejemplo encade. */
  char Sub_cad[30];         /* Contiene la entrada a analizar */
  char Sub_cad2[30];        /* Cons. entrada sacara el Campo a proyectar */
  char Camp_op[10];         /* Cont. Campo a proyectar */
  char Elemento[5];         /* Cont. elemento encadenara entradas */
  char Cadena[Tam_entrada];
  char Relacion =  = ;
  char Bandera, Banderita;
  char Band_nodo[5];  /* Indica si se formo Nodo correspon. */
  int Posi;
  int Apun_operan[5]; /* Ap. ult. caract. puesto en Operandos */
  struct Nodo_arbol *Ap_nodo[5];  /* Apunt. nodo recien creado */
  struct Nodo_arbol *Crea_nodo();
  char Busca_ca();

  for(j = 0; j <= Num_line; j++)
    {
    Band_nodo[j] =  n ;
    Apun_operan[j] = 0;
    } /*Fin for j*/
  strcpy(Cadena, Cad);
  Bandera = Compara(Cadena, "].", &Posi);
  while (Bandera ==  s  && Cadena[0] != '\0')
    {
    Bandera = Busca_ca(Cadena, Sub_cad, "[.");
    Sep_infor(Sub_cad, Camp_op, Elemento, &Relacion);
    for(j = 0; j <= Num_line; j++)
      {
      if(j == i || Nodo_hoja[j].Caja_cond ==  s ) continue;
      strcpy(Tabla, Nodo_hoja[j].Entradas);
      Banderita = Busca_ca(Tabla, Sub_cad2, Elemento);
      if (Banderita ==  s )
        {
        Sep_infor(Sub_cad2, Camp_op, Elemento, &Relacion);
        if (Band_nodo[j] ==  n )
          {
          Ap_nodo[j] = Crea_nodo();
          Ap_nodo[j]->Oper_realizar[0] = 'P' ;
          Ap_nodo[j]->Oper_realizar[1] =  u ;
```

```
        Ap_nodo[j]->Oper_realizar[2] =  \0 ;
        Band_nodo[j] =  s ;
        Enca_nodo(Ap_nodo[j], j, 2);
        if(j > 0 && Ap_nodo[j-1]->Oper_realizar[0] ==  P  &&
                    Ap_nodo[j-1]->Oper_realizar[1] ==  u  &&
                    Ap_nodo[j-1]->Apunta_padre == Ap_nodo[j] )
          {
          Ap_nodo[j] = Ap_nodo[j-1];
          Ap_nodo[j-1]->Apunta_padre = (struct Nodo_arbol *) NULL;
          Apun_operan[j] = Apun_operan[j-1];
          Ap_nodo[j]->Operandos[Apun_operan[j]++] =     ;
          Entrada[j].Apun_ultimo = Ap_nodo[j];
          }  /*Fin if j*/
        for(l = 0; (Ap_nodo[j]->Operandos[Apun_operan[j]] = Camp_op[l])
              !=  \0 ; l++, Apun_operan[j]++);
          } /*Fin if Band_nodo*/
      else
        {
        Ap_nodo[j]->Operandos[Apun_operan[j]++] =     ;
        for(l = 0; (Ap_nodo[j]->Operandos[Apun_operan[j]] = Camp_op[l]
              !=  \0 ; l++, Apun_operan[j]++);
        }   /*RESTAURAR*/
      break;
      } /*Fin if Banderita*/
    } /*Fin for j*/
  Bandera = Compara(Cadena, "I.", &Posi);
  }  /*Fin while*/
}  /*Fin Operand_TR*/
*/
```

```
/* Localiza la funcion previamente que se tenga en la   */
/* solicitud, determinara el campo de la entrada en que */
/* se encuentra.                                        */
/******************************************************/
/*                                                    */
/*            SUBRUTINA FUN_PRE                       */
/*                                                    */
/******************************************************/

Fun_pre()

  {
  int i, j, k, m;
  int k = 999;
  int Posicion;
  char Ban_fun_CC =  n ;
  char Ban_fun =  n ;
  char Bandera;
  char Elemento[15];
  char Cadena[tam_entrada];
  char Sub_cad[30];
  char Cam_oper[10];
  char Funcion[5];
  char Relacion =  = ;
  struct Nodo_arbol *Ap_nodo;
  struct Nodo_arbol *Crea_nodo();
  char Hay_fun();
  char Compara();
  char Busca_ca();

  for (i = 0; i <= Num_line; i++)
    if (Nodo_hoja[i].Caja_cond ==  s )
      {
      Ban_fun_CC = Hay_fun(Nodo_hoja[i].Entradas, &j);
      k = i;
      break;
      } /*Fin if Nodo_hoja*/
  if (Ban_fun_CC ==  s )
    {
    Bandera = Compara(Nodo_hoja[k].Entradas, "E.", &Posicion);
    Posicion += 3;
    for (i = 0; Nodo_hoja[k].Entradas[Posicion] !=    ; i++, Posicion++)
      Elemento[i] = Nodo_hoja[k].Entradas[Posicion];
    Elemento[i] =  \0 ;
    for (i = 0; i <= Num_line; i++)
      {
      if (i == k) continue;
      strcpy(Cadena, Nodo_hoja[i].Entradas);
      Bandera = Busca_ca(Cadena, Sub_cad, Elemento);
      if (Bandera ==  s')
        {
        Sep_infor(Sub_cad, Cam_oper, Elemento, &Relacion);
        k = i;
        break;
```

```
      } /*Fin if Bandera = 's */
    } /*Fin for i*/
  } /*Fin if Ban_fun_CC*/
else
  for (i = 0; i <= Num_line; i++)
    {
    if(Nodo_hoja[i].Caja_cond == 's'
       || Nodo_hoja[i].Caja_resu == 's')
      continue;
    Ban_fun = Hay_fun(Nodo_hoja[i].Entradas, &j);
    if (Ban_fun == 's')
      {
      k = i;
      break;
      } /*Fin if Ban_fun*/
    } /*Fin for i*/
if (Ban_fun == 's' || Ban_fun_CC == 's')
  {
  switch(j)
    {
    case 0: strcpy(Funcion, "CNT");
            break;
    case 1: strcpy(Funcion, "SUM");
            break;
    case 2: strcpy(Funcion, "PRM");
            break;
    case 3: strcpy(Funcion, "MAX");
            break;
    case 4: strcpy(Funcion, "MIN");
            break;
    } /*Fin switch*/
  if (Ban_fun == 's' )
    {
    strcpy(Cadena, Nodo_hoja[k].Entradas);
    Bandera = Busca_ca(Cadena, Sub_cad, Funcion);
    Sep_infor(Sub_cad, Cam_oper, Elemento, &Relacion);
    } /*Fin Ban_fun interno*/
  Ap_nodo = Crea_nodo();
  Ap_nodo->Oper_realizar[0] = 'F' ; Ap_nodo->Oper_realizar[1] = 'p' ;
  Ap_nodo->Oper_realizar[2] = '\0' ;
  for(i = 0; (Ap_nodo->Operandos[i] = Cam_oper[i]) != '\0'; i++);
  Ap_nodo->Operandos[i++] = '.';
  for(m = 0; (Ap_nodo->Operandos[i] = Funcion[m]) != '\0'; i++, m++);
  Enca_nodo(Ap_nodo, k, 2);
  } /*Fin if Ban_fun externo*/
return(k);
} /*Fin Fun_pre*/
/**/
```

```
/* Localizara todos los operandos necesarios para realiz- */
/* zar la la operacion de Seleccion de Grupo.             */
/*                                                        */
/*                SUBRUTINA SEL_GRUP                      */
/*                                                        */

Sel_grup(Line_FF)

int Line_FF;

  {
  struct Nodo_arbol *Ap_nodo, *Ap_aux;
  int i, j, k;
  int Posicion;
  char Band;
  char Condicion[15];
  char Compara();
  struct Nodo_arbol *Crea_nodo();

  Ap_nodo = Crea_nodo();
  Ap_nodo->Oper_realizar[0] =  S ; Ap_nodo->Oper_realizar[1] =  g ;
  Ap_nodo->Oper_realizar[2] = '\0' ;
  Ap_aux = Nodo_hoja[Line_FF].Apun_padre;
  while(Ap_aux->Apunta_padre != (struct Nodo_arbol *) NULL)
    Ap_aux = Ap_aux->Apunta_padre;
  Enca_nodo(Ap_nodo, Line_FF, 2);
  for (i = 0; Ap_aux->Operandos[i] !=  '\0' ; i++)
    Ap_nodo->Operandos[i] = Ap_aux->Operandos[i];
  Ap_nodo->Operandos[i++] =    ;
  for (j = 0; j <= Num_line; j++)
    if(Nodo_hoja[j].Lala_cond ==  s ) break;
  Band = Compara(Nodo_hoja[j].Entradas, "E.", &Posicion);
  Posicion += 3;
  for (; Nodo_hoja[j].Entradas[Posicion] !=    ; Posicion++);
  Posicion++;
  for (k = 0; (Condicion[k] = Nodo_hoja[j].Entradas[Posicion]) != ')';
     Posicion++, k++);
  Condicion[k] = '\0' ;
  for (k = 0; (Ap_nodo->Operandos[i] = Condicion[k]) != '\0' ;
      k++, i++);
  }  /*Fin Sel_grup*/
/**/
```

```
/* Encuentra aquellas relaciones que se une por la opera-*/
/* cion denominada Relacion de equivalencia              */
/*******************************************************/
/*                                                     */
/*              SUBRUTINA RELA_EQUI                    */
/*                                                     */
/*******************************************************/

Rela_equi()

  {
  struct Nodo_arbol *Ap_nodo;
  int i, Par;
  int Posicion;
  char Bandera, bandera2, Band;
  int Ap_opera = 0;
  char Relacion, Relac2;
  char Op_logic =  & ;
  char Contenido;
  char Cadena1[am_entrada], Cadena2[lam_entrada];
  char Sub_cad[35];
  char Elemento[15];
  char Camp_op[10], Camp_op2[10];
  char Compara();
  char Busca_ca();
  struct Nodo_arbol *Crea_nodo();

  for (i = 0; i <= Num_line; i++)
    if (Nodo_hoja[i].Tabla_unib ==  n') break;
  strcpy(Cadena, Nodo_hoja[i].Entradas);
  Bandera = Busca_ca(Cadena, Sub_cad, "E.");
  while (Bandera ==  s')
    {
    Sep_infor(Sub_cad, Camp_op, Elemento, &Relacion);
    for(Par = 0; Par <= Num_line; Par++)
      {
      if (Par == i || Nodo_hoja[Par].Caja_cond ==  s ) continue;
      Bandera2 = Compara(Nodo_hoja[Par].Entradas, Elemento, &Posicion);
     * if (Bandera2 ==  s )
        {
        Ap_nodo = Crea_nodo();
        Ap_nodo->Oper_realizar[0] = 'R ;
        Ap_nodo->Oper_realizar[1] =  e ;
        Ap_nodo->Oper_realizar[2] =  \0';
        Enca_nodo(Ap_nodo, Par, 2);
        Enca_nodo(Ap_nodo, i, 1);
        strcpy(Cadena2, Nodo_hoja[Par].Entradas);
        Band = Busca_ca(Cadena2, Sub_cad, Elemento);
        Sep_infor(Sub_cad, Camp_op2, Elemento, &Relac2);
        if (Relac2 !=  = )
          switch (Relac2)
            {
            case  <': Relacion =  > ;
                      break;
```

```
            case  > : Relacion =  < ;
                      break;
            case    : Relacion = Relacia;
                      break;
            }  /*Fin switch*/
          Pon_operando(ap_nodo, Camp_op, Relacion, Camp_op2, &ap_opera,
                       Op_logic);
          Band = Compara(Nodo_hoja[i].Entradas, "I.", &Posicion);
          if (Band ==  s )
            Contenido =  i ;
          else
            Contenido =  d ;
          Ap_nodo->Operandos[ap_opera++] = Contenido;
          Ap_nodo->Operandos[ap_opera] =  \0 ;
          break;
          }  /*Fin if Bandera2*/
        }  /*Fin for*/
      if (Bandera2 ==  s  || Cadena[0] ==  \0 ) break;
      bandera = Busca_ca(Cadena, Sub_cad, "E.");
      }  /*Fin while*/
    }  /*Fin Rela_cpu*/
  /**/
```

```
/* Realiza la Proyeccion final de los campos cuyo conte  */
/* nido, o resultado de aplicarle una funcion, se desea  */
/* imprimir como resultado. La proyeccion se realiza con */
/* siderandos las cuatro casos que se tienen para pedir  */
/* la impresion de informacion.                          */
/*********************************************************/
/*                                                       */
/*              SUBRUTINA PROY_FINAL                     */
/*                                                       */
/*********************************************************/

Proy_final()

 {
 struct Nodo_arbol *Ap_nodo, *Ap_nodo2;
 int i, j;
 int Posi;
 char Band_IR =  n ;
 char Bandera, Banderita;
 char Relacion =  = ;
 char Band_fun;
 char Band_gru;
 char Operandos[50];
 char Cadena[Tam_entrada];
 char Sub_cad[30];
 char Elemento[15];
 char Camp_op[10];
 char Funcion[4];
 struct Nodo_arbol *Crea_nodo();
 char Compara();
 char busca_ca();
 char Hay_fun();

 for (i = 0; i <= Num_line; i++)
   if (Nodo_hoja[i].Tabla_resul ==  s')
     {
     Band_IR =  s ;
     break;
     }
 if (Band_IR ==  s )
   {
   char Line_anal[Max_tablas];

   for (j = 0; j <= Num_line; j++)
     Line_anal[j] =  n ;
   strcpy(Cadena, Nodo_hoja[i].Entrades);
   Bandera = Compara(Cadena, "I.", &Posi);
   while (Bandera ==  s')
     {
     Bandera = Busca_ca(Cadena, Sub_cad, "I.");
     Sep_infor(Sub_cad, Camp_op, Elemento, &Relacion);
     for (j = 0; j <= Num_line; j++)
       {
       if (j == i) continue;
```

```
      Banderita = Compara(Nodo_hoja[j].Entradas, Elemento, &Posi);
      if (Banderita == 's' )
        {
        if (Line_ana[j] ==  's' ) break;
        Line_ana[j] =  's' ;
        Ap_nodo2 = Entradas[j].Apun_ultimo;
        Ap_nodo = Crea_nodo();
        Enca_nodo(Ap_nodo, j, 2);
        if(j > 0 && Entradas[j-1].Apun_ultimo->Oper_realizar[0] ==  'P' &&
                    Entradas[j-1].Apun_ultimo->Oper_realizar[1] ==  'r' &&
                    Entradas[j-1].Apun_ultimo->Apunta_padre == Ap_nodo )
          {
          Entradas[j].Apun_ultimo = Entradas[j-1].Apun_ultimo;
          Entradas[j-1].Apun_ultimo->Apunta_padre =
                              (struct Nodo_arbol *) NULL;
          Ap_nodo->Hijo_der = (struct Nodo_arbol *) NULL;
          break;
          }  /*Fin if j*/
        strcpy(Ap_nodo->Oper_realizar, "Prv");
        strcpy(Ap_nodo->Operandos, Ap_nodo2->Operandos);
        break;
        }  /*Fin if Banderita*/
      }  /*Fin for j*/
    Bandera = Compara(Cadena, "I.", &Posi);
    }  /*Fin while*/
  }  /*Fin if Band_IR*/
else
  {
  for (i = 0; i <= Num_line; i++)
    {
    if (Nodo_hoja[i].Caja_cond ==  's' ) continue;
    Bandera = Compara(Nodo_hoja[i].Entradas, "I.", &Posi);
    if (Bandera ==  's' )
      {
      strcpy(Cadena, Nodo_hoja[i].Entradas);
      Operandos[0] =  '\0' ;
      do
        {
        Bandera = Busca_ca(Cadena, Sub_cad, "I.");
        Band_fun = Hay_fun(Sub_cad, &j);
        Band_gru = Compara(Sub_cad, "A.", &Posi);
        if (Band_fun ==  's' )
          switch(j)
            {
            case 0: strcpy(Funcion, "UNI");
                    break;
            case 1: strcpy(Funcion, "SUM");
                    break;
            case 2: strcpy(Funcion, "PRM");
                    break;
            case 3: strcpy(Funcion, "MAX");
                    break;
            case 4: strcpy(Funcion, "MIN");
                    break;
```

```
            }  /*Fin switch*/
          Sep_infor(Sub_cad, Camp_op, Elemento, &Relacion);
          strcat(Operandos, Camp_op);
          strcat(Operandos, " ");
          if (Band_fun ==  s )
            strcat(Operandos, Funcion);
          else if (Band_gru ==  s )
                strcat(Operandos, "A");
          strcat(Operandos, " \n");
          }  /*Fin do*/
        while ((Bandera = Compara(Cadena, "1.", &Pos)) == 's );
        break;
        }  /*Fin if Bandera*/
      }  /*Fin for*/
    Ap_nodo = Crea_nodo();
    strcpy(Ap_nodo->Oper_realizar, "Pr\0");
    strcpy(Ap_nodo->Operandos, Operandos);
    Inca_nodo(Ap_nodo, i, 2);
    }  /*Fin else if Band_IR*/
  }  /*Fin Proy_final*/
/**/
```

```
/* Este Archivo contiene solo funciones que se encargaran de Parsear */
/* Palabras Claves que implican funciones de No-grupo.               */
/* Funcion maestra que llamara a otras para que en total parsean las */
/* operaciones que no implican grupos                                */
char *strcpy();
/*****************************************************************/
/*                                                               */
/*              SUBRUTINA PAR_NO_GR                              */
/*                                                               */
/*****************************************************************/

Par_no_gr()

  {
  int i;

  Seleccion();
  Join();
  }/*END Par_no_Gr*/
```

```
/* Se localiza todas las operaciones de seleccion. Hara todas las opera- */
/* ciones implicadas con la seleccion                                    */
/*************************************************************/
/*                                                           */
/*            SUBRUTINA SELECCION                            */
/*                                                           */
/*************************************************************/

Seleccion()

 {
 char Cadena[Tam_entrada];  /* Entradas de una linea a analizar */
 char Sub_cad[25];          /* Contiene la entrada a analizar */
 char Sub_busca[6];         /* Cont. cadena que senala la entr. analiz */
 char Banderita;           /* Indica si encontro una entrada de interes */
 char Camp_op[10];          /* Campo al que se le hara operacion */
 char Elemento [15];        /* Elemento Constante de Operacion */
 int i;                     /* Variable de trabajo */
 char Compara();
 char Busca_ca();
 struct Nodo_arbol *Crea_nodo();

 for (i = 0; i <= Num_line; i++)
   {
   strcpy(Cadena, Nodo_hoja[i].Entradas);
   if (Nodo_hoja[i].Caja_cond ==  's' )
     {
     int Posi;

     strcpy(Sub_busca, "Todo.");
     Banderita = Compara(Cadena, Sub_busca, &Posi);
     if (Banderita ==  's' ) continue;
     Bus_c_CL(i);
     continue;              /* Si linea es C. condicion no la analiza */
     }
   if (Nodo_hoja[i].Tabla_resul ==  's' ) continue;
   strcpy(Sub_busca, "C.");
   Banderita = Busca_ca(Cadena, Sub_cad, Sub_busca);
   if (Banderita ==  's' )
     {
     int Ap_opera = 0;  /* Dentro del campo de operados del nodo */
                        /* Represen. apun. ultimo caract. escrit. */
     struct Nodo_arbol *Ap_nodo;
     char Relacion =  = ;
     char Op_logico =  & ;

     Ap_nodo = Crea_nodo();
     Entradas[i].Apun_ultimo = Nodo_hoja[i].Apun_padre = Ap_nodo;
     Ap_nodo -> Apun_hod = i;
     Sep_infor(Sub_cad, Camp_op, Elemento, &Relacion);
     Fon_operan(Ap_nodo, Camp_op, Relacion, Elemento, &Ap_opera,
                Op_logico);
     while (Cadena[0] != '\0'  && Banderita == 's')
       {
```

```
        Banderita = Busca_ca(Cadena, Sub_cad, Sub_busca);
        if (Banderita ==  s )
          {
          Sep_inter(Sub_cad, Camp_op, Elemento, &Relacion);
          Pon_operan(Ap_nodo, Camp_op, Relacion, Elemento,&Ap_opera,
                     Op_logico);
          }    /*Fin if Banderita*/
        }    /*Fin while*/
      Ap_nodo -> Operandos[Ap_opera] = '\0' ;
      Ap_nodo -> Oper_realizar[0] =  S ;
      }  /*Fin if banderita*/
    }    /*Fin for*/
  }    /*Fin Seleccion*/
/**/

/* En caso de tener Caja Condicion en solicitud y esta no */
/* tenga el operador de grupo  Todo , busco por alguna   */
/* entrada tenga el mismo elemento ejemplo de la caja    */
/**********************************************************/
/*                                                        */
/*            SUBRUTINA BUS_C_CC                          */
/*                                                        */
/**********************************************************/

Bus_c_CC()

int i;

 {
 int j, k;
 char Banderita;
 char Sub_cad[20];
 char Sub_busca[10];
 char Cadena[Tam_entrada];
 int Posicion;
 char Relacion;
 char Camp_op[10];
 char Elemento[15];
 struct Nodo_arbol *Ap_nodo;
 char Compara();
 struct Nodo_arbol *Crea_nodo();

 for (j = 4, k = 0; (Sub_busca[k] = Nodo_hoja[i].Entradas[j]) !=  = ;
      j++, k++);
 Sub_busca[--k] =  \0 ;
 for (j = 0; j <= Num_line; j++)
   {
   if (j == i)  continue;
   strcpy(Cadena, Nodo_hoja[j].Entradas);
   Banderita = Compara(Cadena, Sub_busca, &Posicion);
   if (Banderita ==  n ) continue;
   Ap_nodo = Crea_nodo();
   Entradas[i].Apun_ultimo = Nodo_hoja[j].Apun_padre = Ap_nodo;
   Sep_ent(Cadena, Sub_cad, Posicion);
```

```
    Sep_inter(Sub_cad, Camp_op, Elemento, &Relacion);
    Fon_op_CC(i, k += 8, Ap_nodo, Camp_op);
    Ap_nodo -> Oper_realizar[0] =  S ;
    break;
    )
  }  /*Fin Bus_c_CC*/
/**/
```

```
/* Pone los operandos de una Operacion de Seleccion cuan*/
/* do estos estan dados en una Caja de Condicion        */
/*******************************************************/
/*                                                     */
/*            SUBRUTINA PON_OP_CC                      */
/*                                                     */
/*******************************************************/

Pon_op_CC(i, Ap, Ap_nodo, Camp_op)

#define CC Nodo_hoja[i]

int i, Ap;
struct Nodo_arbol *Ap_nodo;
char *Camp_op;

  {
  char Relacion;
  char Elemento[15];
  char Op_logico =    ;
  int Ap_opera = 0;
  int k;

  while (CC.Entradas[Ap] !=  )  && CC.Entradas[Ap] !=  '\0' )
    {
    if (CC.Entradas[Ap] ==  <  || CC.Entradas[Ap] == >
        || CC.Entradas[Ap] ==  ^')
      {
      Relacion = CC.Entradas[Ap];
      Ap ++;
      }
    else
      Relacion =  = ;
    for (k = 0; CC.Entradas[Ap] !=  ' ' && CC.Entradas[Ap] !=  ) ;
        Ap++, k++)
      Elemento[k] = CC.Entradas[Ap];
    Elemento[k] = '\0' ;
    Pon_operan(Ap_nodo, Camp_op, Relacion, Elemento, &Ap_opera,
               Op_logico);
    if (CC.Entradas[Ap] ==     )
      {
      Op_logico = CC.Entradas[++Ap];
      Ap += 2;
      }
    }
  } /*Fin Pon_op_CC*/
#undef CC
/**/
```

```
/* Localiza todas las operaciones de Join que aparescan. */
/* Hara todas las operaciones que ello implica           */
/*********************************************************/
/*                                                       */
/*            SUBRUTINA JOIN                             */
/*                                                       */
/*********************************************************/

Join()

 {
 int i, j;
 char Cadena[Tam_entrada]; /* Entradas de una linea a analizar */
 char Sub_cad[25];         /* Contiene la entrada a analizar */
 char Sub_busca[5];        /* Cont. cadena que busca la entr. analiz. */
 char Banderita;           /* Indica si se encontro entrada de interes */
 char Camp_op[10];         /* Campo al que se le hara la operacion */
 char Elemento[15];        /* Elemento ejemplo de la operacion */
 char Band;
 int Line_par;
 struct Nodo_arbol *Crea_nodo();
 char Compara();
 char Busca_ca();
 char Ana_antes();

 for (i = 0; i <= Num_line; i++)
   {
   if (Nodo_hoja[i].Tabla_resul ==  s ) continue; /* Si IR no analiza */
   if (Nodo_hoja[i].Caja_cond ==  s ) continue;  /* Si CC no analiz. */
   if (Nodo_hoja[i].Tabla_unib ==  n ) continue; /* Si No unib. no ana. */
   strcpy(Cadena, Nodo_hoja[i].Entradas);
   strcpy(Sub_busca, "E.");
   Banderita = Busca_ca(Cadena, Sub_cad, Sub_busca);
   while (Banderita ==  s )
     {
     char Cadnita[6];
     int Ap_opera = 0;  /* Dentro del campo operandos del nodo */
                        /* apunta al ultimo caracter escrito   */
     struct Nodo_arbol *Ap_nodo;
     char Relacion =  = ;
     char Op_logico =  & ;
     int Posi;

     for (j = 0; j <= 2; j++)
       {
       switch(j)
         {
         case 0: strcpy(Cadnita, "Nodo.");
                 Band = Compara(Sub_cad, Cadnita, &Posi);
                 break;
         case 1: Sep_infor(Sub_cad, Camp_op, Elemento, &Relacion);
                 Band = Ana_antes(i - 1, Elemento);
                 break;
         case 2: Line_par = Sig_ejem(Elemento, i + 1);
```

```
                        if(Line_par != -1 &&
                           (Nodo_hoja[Line_par].Caja_cond ==  s  ||
                            Nodo_hoja[Line_par].tabla_resul ==  s ))
                          Line_par = -1;
                    }                          /*Fin switch*/
                  if (Band ==  s  || (Line_par == -1 && j == 2)) break;
                  }                           /*Fin for  j */
                if (Band ==  n  && Line_par != -1 &&
                   Nodo_hoja[Line_par].tabla_unib ==  s )
                  {
                  char Cadena2[fam_entrada];
                  char Camp_op2[10];
                  char Relaci2 =  = ;

                  Ap_nodo = Crea_nodo();
                  Ap_nodo -> Oper_realizar[0] =  J ;
                  Enca_nodo(Ap_nodo, j, 2);
                  Enca_nodo(Ap_nodo, Line_par, 1);
                  strcpy(Cadena2, Nodo_hoja[Line_par].Entradas);
                  Banderita = Busca_ca(Cadena2, Sub_cad, Elemento);
                  Sep_infor(Sub_cad, Camp_op2, Elemento, &Relaci2);
                  if (Relacion !=  = )
                    switch(Relacion)
                      {
                      case  <: Relaci2 =  > ;
                               break;
                      case  > : Relaci2 =  < ;
                               break;
                      caso  =: Relaci2 = Relacion;
                               break;
                      }
                  Pon_operan(Ap_nodo, Camp_op2, Relaci2, Camp_op, &Ap_opera,
                             Op_logico);
                  }                            /*Fin if Band*/
                if (Cadena[0] == '\0' ) break;
                Banderita = Busca_ca(Cadena, Sub_cad, Sub_busca);
                }                             /*Fin while*/
              }                              /*Fin del for */
            }    /*Fin Join*/
          /**/
```

```
/* ------------------------------------------------------------ */

/* RECORRIDO DEL ARBOL DE PARSER DINAMICO GENERADO */


/* Analiza el primer nodo de la rama 0 del arbol de   */
/* Parser.                                            */
/******************************************************/
/*                                                    */
/*              PROCEDIMIENTO PRIM_NODO               */
/*                                                    */
/******************************************************/

Prim_nodo(Operacion, Operandos, Arch_in)

char *Operacion, *Operandos, *Arch_in;

  {
  int Hijo;
  char Arch_aux[25];
  int i;

  Ramas_reco[Rama_reco] =  s ;
  Nodo_anali[Num_sub] = Nodo_hoja[Rama_reco].Apun_padre;
  for(i = 0; (Arch_in[i] = Nodo_hoja[Rama_reco].Nom_tabla[i]) !=  \0 ; i++);
  strcpy(Operacion, Nodo_hoja[Rama_reco].Apun_padre->Oper_realizar);
  strcpy(Operandos, Nodo_hoja[Rama_reco].Apun_padre->Operandos);
  if (Operacion[0] ==  J  || (Operacion[0] ==  R  && Operacion[1] ==  e ))
    {
    if (Pendiente ==  n )
      {
      Pendiente =  s ;
      strcpy(Oper_pendi,Operacion);
      strcpy(Operan_pendi, Operandos);
      strcpy(Archivo_pendi, Arch_in);
      Nodo_pendi = Nodo_anali[Num_sub];
      if(Operacion[0] ==  J )
        Hijo = 1;  /*Pareja es hijo izquierdo*/
      else
        Hijo = 2;  /*Pareja es hijo derecho*/
      strcpy(Operacion, "NN");  /* NN  indica clipper nodo Pendiente*/
      Rama_pareja[Hijo];
      }  /*Fin if Pendiente*/
    else
      {
      Pendiente =  n ;
      strcpy(Operacion, Oper_pendi);
      strcpy(Operandos, Operan_pendi);
      Nodo_anali[Num_sub] = Nodo_pendi;
      if(Operacion[0] ==  J )
        {
        strcat(Arch_in, "-");
        strcat(Arch_in, Archivo_pendi);
        }  /*Fin If Operacion*/
```

```
      else
         {
         strcat(archivo_pendi, "-");
         strcat(archivo_pendi, Arch_in);
         strcpy(Arch_in, Archivo_pendi);
         }  /*Fin else Operacion*/
      }  /*Fin else Pendiente*/
    }  /*Fin if Operacion*/
  } /*Fin Primer_nodo*/
/**/


/* Continua el recorrido del Arbol, tratando de visitar */
/* todos los nodos que constituyen el arbol. Regresa   */
/* una bandera que indica si hay mas nodos por reco-   */
/* rrer en una rama.                                   */
/*******************************************************/
/*                                                     */
/*              PROCEDIMIENTO RECORRIDO                */
/*                                                     */
/*******************************************************/

Recorrido(Operacion, Operandos, Arch_in, Mas_ramas, Bandera)

char *Operacion, *Operandos, *Arch_in;
char *Mas_ramas;            /* Nos dice si hay mas ramas a recorrer */
char *Bandera;

  {
  char N_arch_in[2], Un_arch_in[2];
  int i;
  int Hijo;

  Mas_ramas[1] = '\0';  N_arch_in[1] = '\0';  Un_arch_in[1] = '\0';
  Mas_ramas[0] =  s ;
  if (Nodo_anali[Num_sub].Apunta_padre == (struct Nodo_arbol *) NULL)
    {                                  /*Se recorrio toda la rama*/
    for (i = 0; i <= Num_line; i++)
      {
      if (Ramas_reco[i] == 'n'  && Nodo_hoja[i].Caja_cond ==  n  &&
                                   Nodo_hoja[i].Tabla_resul ==  n')
        {
        Rama_reco = i;
        Mas_ramas[0] =  s ;
        Num_sub = Num_sub + 1;
        break;
        }  /*Fin if Ramas_reco*/
      }  /*Fin for i*/
    if (i > Num_line || Num_sub > 2)
      Mas_ramas[0] =  n';  /*Se termino recorrido del arbol*/
    Bandera[0] =  n ;
    return;
    }  /*Fin if Nodo_anali*/
  Bandera[0] =  s';
```

```
strcpy(Arch_in, Nodo_anali[Num_sub]->Arch_sal);
switch(Nodo_anali[Num_sub]->Num_ar_s)
  {
  case 1:  N_arch_in[0] =  1 ;
           break;
  case 2:  N_arch_in[0] =  2 ;
           break;
  case 3:  N_arch_in[0] =  3 ;
           break;
  default: N_arch_in[0] =  4 ;
  }  /*Fin switch*/
if(N_arch_in[0] ==  1 )
  Un_arch_in[0] =  s ;
else
  Un_arch_in[0] =  n ;
Nodo_anali[Num_sub] = Nodo_anali[Num_sub]->Apunta_padre;
strcpy(Operacion, Nodo_anali[Num_sub]->Oper_realizar);
strcpy(Operandos, Nodo_anali[Num_sub]->Operandos);
if(Operacion[0] ==  J' || (Operacion[0] ==  R  && Operacion[1] ==  e ))
  if(Pendiente ==  n )
    {
    Pendiente =  s ;
    strcpy(Oper_pendi, Operacion);
    strcpy(Operan_pendi, Operandos);
    strcpy(Archivo_pendi, Arch_in);
    Nodo_pendi = Nodo_anali[Num_sub];
    if(Operacion[0] ==  J )
      Hijo = 1;  /*Pareja es hijo izquierdo*/
    else
      Hijo = 2;  /*Pareja es hijo derecho*/
    strcpy(Operacion, "NN");
    Rama_pareja(Hijo);
    Bandera[0] =  n ;
    return;
    }  /*Fin if Pendiente*/
  else
    {
    Pendiente =  n ;
    strcpy(Operacion, Oper_pendi);
    strcpy(Operandos, Operan_pendi);
    Nodo_anali[Num_sub] = Nodo_pendi;
    if(Operacion[0] ==  J )
      {
      strcat(Arch_in, "-");
      strcat(Arch_in, Archivo_pendi);
      }  /*Fin if Operacion*/
    else
      {
      strcat(Archivo_pendi, "-");
      strcat(Archivo_pendi, Arch_in);
      strcpy(Arch_in, Archivo_pendi);
      }  /*Fin else Operacion*/
    }  /*Fin else Pendiente*/
return;
```

```
  }  /*Fin Recorrido*/
/**/


/* Recive todos los parametros que envie la rutina de  */
/* Clipper, despues de realizar alguna Operacion sobre */
/* la Base de Datos.                                   */
/*******************************************************/
/*                                                     */
/*              PROCEDIMIENTO REGRE_PARA               */
/*                                                     */
/*******************************************************/

Regre_para(Arch_out, N_arch_sal, Un_arch_sal)

char *Arch_out, *N_arch_sal, *Un_arch_sal;

  {
  strcpy(Nodo_anali[Num_sub]->Arch_sal, Arch_out);
  Nodo_anali[Num_sub]->Out_one_file = Un_arch_sal[0];
  switch(N_arch_sal[0])
    {
    case  1 :  Nodo_anali[Num_sub]->Num_ar_s = 1;
               break;
    case  2':  Nodo_anali[Num_sub]->Num_ar_s = 2;
               break;
    case  3 :  Nodo_anali[Num_sub]->Num_ar_s = 3;
               break;
    default:   Nodo_anali[Num_sub]->Num_ar_s = 4;
    }  /*Fin switch*/
  }  /*Fin Regre_para*/
/**/


/* Localiza al Nodo Hoja de la rama que es pareja de   */
/* aquelle que entra a un Nodo con una operacion Bina- */
/* ria.                                                */
/*******************************************************/
/*                                                     */
/*              PROCEDIMIENTO RAMA_PAREJA              */
/*                                                     */
/*******************************************************/

Rama_pareja(Hijo)

int Hijo;

  {
  struct Nodo_arbol *Apunta;

  Apunta = Nodo_pend1;
  if(Hijo == 2 && Apunta->Apun_hod != 999)
    {
    Rama_reco = Apunta->Apun_hod;
```

```
    return;
    }  /*Fin if Hijo*/
  if(Hijo == 1 && Apunta->Apun_hoi != 999)
    {
    Rama_reco = Apunta->Apun_hoi;
    return;
    }  /*Fin if Hijo*/
  if(Hijo == 2)
    Apunta = Apunta->Hijo_der;
  else
    Apunta = Apunta->Hijo_izq;
  while(Apunta->Hijo_der != (struct Nodo_arboi *) NULL)
    Apunta = Apunta->Hijo_der;  /*Localiza Nodo Hoja*/
  Rama_reco = Apunta->Apun_hod;
  }  /*Fin Rama_pareja*/
/**/


/* Es el Procedimiento maestro de la etapa de generacion  */
/* de los esqueletos que se usaran en la solicitud.       */
/*********************************************************/
/*                                                       */
/*                  PROCEDIMIENTO ESQUELET               */
/*                                                       */
/*********************************************************/

Esquelet(Cad_1, Cad_2, Cad_3, Cad_4)

char *Cad_1, *Cad_2, *Cad_3, *Cad_4;

  {
  int i;
  int Linea;
  int Num_cad;
  int Diferencia = 5;
  int Tipo_esquel5];
  char Band =  s ;
  char Cadena[40];

  Lim_pan();
  Linea = 2;
  for (Num_cad = 4;;Num_cad--)
    {
    switch(Num_cad)
      {
      case 4:  if(Cad_4[0] !=  0 ) Band =  n';
               break;
      case 3:  if(Cad_3[0] !=  0 ) Band =  n';
               break;
      case 2:  if(Cad_2[0] !=  0') Band =  n ;
               break;
      default: Band =  n';
      }  /*Fin switch*/
    if(Band ==  n ) break;
```

```
    }  /*Fin for Num_cad*/
  Tipo_esque[0] = Num_cad;
  for(i = 1; i <= Num_cad; i++)
    {
    switch(i)
      {
      case 1: strcpy(Cadena, Cad_1);
              break;
      case 2: strcpy(Cadena, Cad_2);
              break;
      case 3: strcpy(Cadena, Cad_3);
              break;
      case 4: strcpy(Cadena, Cad_4);
      }  /*Fin switch i*/
    if (Cadena[0] ==  0 ) continue;
    switch(Cadena[0])
      {
      case  2':Caja_Cond(Linea);
               Tipo_esque[i] = 2;
               break;
      case '3' : Tabla_Re(Linea);
                 Tipo_esque[i] = 3;
                 break;
      default : Relacion(Linea, Cadena);
                Tipo_esque[i] = 4;
      }  /*Fin switch Cadena*/
    Linea = Linea + Diferencia;
    }  /*Fin for i*/
  Llena_esque(Tipo_esque);
  Lee_esque(Tipo_esque);
  Num_line = Num_cad - 1;
  }  /*Fin esquelet*/
/**/


/* Limpia la Pantalla y activa la primer pagina de esta    */
/*********************************************************/
/*                                                       */
/*            PROCEDIMIENTO LIM_PAN                      */
/*                                                       */
/*********************************************************/

Lim_pan()
  {
  scclear();
  scpage(0);
  } /*Fin Lim_pan*/
/**/


/* Dibuja el esqueleto de una Caja de Condicion.         */
/*********************************************************/
/*                                                       */
/*                PROCEDIMIENTO CAJA_COND                */
```

```
}
/*                                                                    */
/**********************************************************************/

Caja_cond(Linea)

int Linea;

  {
  char *Encabezado;

  Encabezado = "CAJA DE CONDICION";
  sccurset(Linea, 19);
  scwrite((char)0xC9, 1);
  sccurset(Linea, 20);
  scwrite((char)0xCD, 40);
  sccurset(Linea, 60);
  scwrite((char)0xBB, 1);

  sccurset(Linea+1, 19);
  scwrite((char)0xBA, 1);
  sccurset(Linea+1, 60);
  scwrite((char)0xBA, 1);

  sccurset(Linea+2, 19);
  scwrite((char)0xCC, 1);
  sccurset(Linea+2, 20);
  scwrite((char)0xCD, 40);
  sccurset(Linea+2, 60);
  scwrite((char)0xB9, 1);

  sccurset(Linea+3, 19);
  scwrite((char)0xBA, 1);
  sccurset(Linea+3, 60);
  scwrite((char)0xBA, 1);

  sccurset(Linea+4, 19);
  scwrite((char)0xBA, 1);
  sccurset(Linea+4, 60);
  scwrite((char)0xBA, 1);

  scdspmsg(Linea+1, 31, 7, 0, Encabezado);
  }  /*fin Caja_cond*/
'**/


/* Llama al procedimiento que dibuja un esqueleto, que vi-*/
'* ene a ser el de una Tabla de Resultados.               */
'**********************************************************/
'*                                                        */
/*                  PROCEDIMIENTO TABLA_RE                */
/*                                                        */
/**********************************************************/

Tabla_re(Linea)
```

```
int Linea;

 {
 Es_Re_IR(Linea);
 scdspmsg(Linea + 1, 4, 7, 0, "IR");
 }  /*Fin Tabla_re*/
/**/


/* Llama al procedimiento que dibuja el esqueleto que vie-*/
/* ne a ser el de una Relacion.                            */
/*********************************************************/
/*                                                       */
/*                  PROCEDIMIENTO RELACION               */
/*                                                       */
/*********************************************************/

Relacion(Linea, Cadena)

int Linea;
char *Cadena;

 {
 int i, ap_1;
 char Letrero[12];
 char Sepa;

 ap_1 = 0;
 Letrero[11] = '\0';
 Es_Re_IR(Linea);
 ap_1 = Busc_sep(Cadena,  | );
 Sep_campo(Cadena, Letrero, ap_1);
 Col_letrero(Letrero, Linea+1, 3, 8);
 for(i = 1; i <= 3; i++)
   {
   if(Cadena[0] ==  |  || Cadena[0] ==  &  || Cadena[0] ==  \0 ) break;
   switch(i)
     {
     case 1:
     case 2:
     case 3: Sepa =  | ;
             break;
     case 4:       Sepa =  & ;
     }  /*Fin switch*/
   ap_1 = busc_sep(Cadena, Sepa);
   Sep_campo(Cadena, Letrero, ap_1);
   Col_letrero(Letrero, Linea+1, 21 * (i - 1) + 19, 10);
   }  /*Fin for i*/
 }  /*Fin Relacion*/
/**/


/* Dibuja un esqueleto, que puede ser el de una caja de   */
```

```
/* Condicion, o el de una tabla de resultados.                    */
/*                                                                */
/*                    PROCEDIMIENTO ES_RE_IR                      */
/*                                                                */
/*                                                                */

Es_Re_IR(Linea)

int Linea;

  {
  int i;

  scCursetCLinea, 5);
  scWrite((char)0xCD, 10);

  scCursetCLinea+2, 5);
  scWrite((char)0xCD, 10);

  for(i = 0; i < 3; i++)
    {
    scCursetCLinea, 13 + (i * 21));
    scWrite((char)0xCB, 1);
    scCursetCLinea, 14 + (i * 21));
    scWrite((char)0xCD, 20);

    scCursetCLinea+1, 13 + (i * 21));
    scWrite((char)0xBA, 1);

    scCursetCLinea+2, 13 + (i * 21));
    scWrite((char)0xCE, 1);
    scCursetCLinea+2, 14 + (i * 21));
    scWrite((char)0xCD, 20);

    scCursetCLinea+3, 13 + (i * 21));
    scWrite((char)0xBA, 1);
    }  /*Fin for*/
  scCursetCLinea, 76);
  scWrite((char)0xBB, 1);

  scCursetCLinea+1, 76);
  scWrite((char)0xBA, 1);

  scCursetCLinea+2, 76);
  scWrite((char)0xB9, 1);

  scCursetCLinea+3, 76);
  scWrite((char)0xBA, 1);
  }  /*Fin Es_Re_IR*/
/**/


/* Tiene la misma funcion que  Compara  del programa que  */
```

```
/* genera el arbol de Parser. Da como resultado la posi-  */
/* cion, dentro  Cadena , en la que se encuentra  Patron .*/
/*****************************************************/
/*                                                   */
/*                PROCEDIMIENTO BUSC_SEP             */
/*                                                   */
/*****************************************************/

Busc_sep(Cadena, Separador)

char *Cadena, Separador;

  {
  int Lim, k;

  Lim = strlen(Cadena);
  for (k = 0; k <= Lim && (Cadena[k] != Separador); k++);
  return(k);
  }  /*Fin Busc_sep*/
/**/


/* Separa la subcadena, que sera cabeza de columna, de la */
/* Cadena , y la elimina de esta ultima.                  */
/*********************************************************/
/*                                                       */
/*                PROCEDIMIENTO SEP_CAMPO                */
/*                                                       */
/*********************************************************/

Sep_campo(Cadena, Letrero, apunta)

char *Cadena, *Letrero;
int apunta;

  {
  int i, j, k;
  char Cad[40];

  for(i = 0; i <= apunta-1; i++)
    Letrero[i] = Cadena[i];
  Letrero[i] =  \0 ;
  k = strlen(Cadena); Cadena[++k] =  \0 ;
  strcpy(Cad, Cadena);
  if (Cad[apunta] ==  & )
    {
    Cadena[0] = '\0' ;
    } /*Fin if*/
  else
    {
    for(i = apunta+1, j = 0; Cad[i] !=  &'; i++, j++)
      Cadena[j] = Cad[i];
    Cadena[j] = Cad[i];
    Cadena[++j] =  \0 ;
```

```
    ) /*Fin else*/
  } /*Fin Sep_campo*/
/**/


/* Presenta un letrero en Pantalla, apartir de una deter- *
/* minada linea y columna, de una determinada longitud.   */
/*********************************************************/
/*                                                       */
/*                  PROCEDIMIENTO COL_LETRERO            */
/*                                                       */
/*********************************************************/

Col_letrero(Letrero, Linea, Columna, Num_caracter)

char *Letrero;
int Linea, Columna, Num_caracter;

  {
  int i;

  for(i = 0; (i < Num_caracter) && Letrero[i] != '\0' ; i++)
    {
    scurset(Linea, Columna + i);
    scwrite(Letrero[i], 1);
    }
  } /*Fin Col_letrero*/
/**/


/* Permite al usuario llenar los campos correspondientes  */
/* de las Tablas que usara para su solicitud. El usuario  */
/* no podra navegar a su antojo por la pantalla.          */
/**********************************************************/
/*                                                        */
/*                  PROCEDIMIENTO LLENA_ESQUE             */
/*                                                        */
/**********************************************************/

Llena_esque(Tablas)

int *Tablas;

  {
  int i;
  int Linea = 2;        /* Linea en que escribira usuario sus letreros */
  int Diferencia = 5;   /* Incremento en lineas de escritura */

  for(i = 1; i <= Tablas[0]; i++)
    {
    int k, frente, fondo;
    char Caracter;
    char scread();
```

```
    switch(Tablas[i])
      {
      case 2:  Llena_usu(Linea + 3, 20, 1, 0);
               break;
      case 3:  Llena_usu(Linea + 1, 14, 3, 21); /* Pone cabeza columnas *
               for(k = 3; k > 1; k--)
                 {
                 scursel(Linea + 1, k * 21 - 7);
                 Caracter = scread(&frente, &fondo);
                 if(Caracter !=    ) break;
                 }  /*Determina num. campos en la tabla */
               Llena_usu(Linea + 3, 14, k, 21); /* Llena columnas */
               break;
      default: for(k = 3; k > 1; k--)
                 {
                 scursel(Linea + 1, (k - 1) * 21 + 19);
                 Caracter = scread(&frente, &fondo);
                 if (Caracter !=    ) break;
                 }  /* Determina num. campos de esta relacion */
               Llena_usu(Linea + 3, 3, 1, 0);
               Llena_usu(Linea + 3, 14, k, 21);
      }  /*Fin switch*/
    Linea = Linea + Diferencia;
    }  /*Fin for i*/
  }  /*Fin Llena_esque*/
/**/


/* Ubicara al cursor en la posicion correspondiente, de a-*/
/* cuerdo al tipo y numero de tabla, para que el usuario  */
/* la llene.                                              */
/*********************************************************/
/*                                                       */
/*                  PROCEDIMIENTO LLENA_USU              */
/*                                                       */
/*********************************************************/

Llena_usu(Linea, Columna, Num_letrero, Desplazamiento)

int Linea;          /* Linea en la que escribira el letrero */
int Columna;        /* Columna apartir de la que escribira letreros */
int Num_letrero;    /* Numero de letreros a escribir en la linea */
int Desplazamiento; /* Separacion entre letreros, si son mas de uno */

  {
  int i, j;
  char Caracter;  /* Caracter recien escrito */

  for (i = 1; i <= Num_letrero; i++)
    {
    scursel(Linea, Columna);
    for(j = 0; (Caracter = getch()) != (015)  &&  Caracter != (033); j++)
      {
      scwrite(Caracter, 1);
```

```
      sccurset(Linea, Columna + j + 1);
      }  /* Fin for j*/
    if(Caracter == (033)) break;
    Columna = Columna + Desplazamiento;
    }  /* Fin for i */
  }  /*Fin Lleva_usu*/
/**/


/* De la forma gratica de la solicitud efectua las lectur- */
/* ras necesarias, de esta, para obtener su forma lineal. */
/*****************************************************/
/*                                                   */
/*                  PROCEDIMIENTO LEE_ESQUE          */
/*                                                   */
/*****************************************************/

Lee_esque(Tablas)

int *Tablas;

  {
  int i;
  int Linea = 2;
  int Diferencia = 5;
  char scread();

  for(i = 1; i <= Tablas[0]; i++)
    {
    int k, j, Frente, Fondo;
    int Ap;
    char Caracter;

    switch(Tablas[i])
      {
      case 2:  strcpy(Lineas[i - 1], "Caja Condicion (");
               Ap = 16;
               Lee_linea(Linea + 3, 20, 40, Lineas[i - 1], &Ap);
               Lineas[i - 1][Ap++] =  ) ;  Lineas[i - 1][Ap] =  \0 ;
               break;
      case 3:  strcpy(Lineas[i - 1], "Tabla Resulta  (");
               Ap = 16;
               for(k = 3; k > 1; k--)
                 {
                 sccurset(Linea + 1, k * 21 - 7);
                 Caracter = scread(&Frente, &Fondo);
                 if(Caracter !=    ) break;
                 }  /*Fin for k*/  /* Calcula num. campos en la Tabla */
               for (j = 1; j <= k; j++)
                 {
                 Lee_linea(Linea + 1, 21*j-7, 20, Lineas[i - 1], &Ap);
                 Lineas[i - 1][Ap++] =  : ;  Lineas[i - 1][Ap++] =    ;
                 Lee_linea(Linea + 3, 21*j-7, 20, Lineas[i - 1], &Ap);
                 Lineas[i - 1][Ap++] =  , ;  Lineas[i - 1][Ap++] =    ;
```

```
                } /*Fin for j*/
              Ap -= 2;
              Lineasli - ]][Ap++] =  ) ;  Lineasli - ]][Ap] =  \0 ;
              break;
      default: sccurset(Linea+3, 3);
              Caracter = scread(&Frente, &Fondo);
              if (Caracter !=    )
                {
                Lineasli - ]][0] =  ' ;
                Lineasli - ]][1] = '  ;
                Ap = 2;
                }
              else
                Ap = 0;
              Lee_linea(Linea+1, 3, 8, Lineasli - 1], &Ap);
              Lineasli - 1][Ap++] =    ;  Lineasli - 1][Ap++] =  ( ;
              for(k = 3; k > 1; k--)
                {
                sccurset(Linea + 1, (k - 1) * 21 + 1);
                Caracter = scread(&Frente, &Fondo);
                if(Caracter !=    ) break;
                }  /*Fin for k*/  /* Calcula num. campos en la hela */
              for(j = 1; j <= k; j++)
                {
                sccurset(Linea + 3, 21 * j - 7);
                Caracter = scread(&Frente, &Fondo);
                if(Caracter ==    ) continue; /* Columna blanca salta */
                Lee_linea(Linea + 1, 21*(j-1)+19, 12, Lineasli - 1], &Ap);
                Lineasli - ]][Ap++] =  : ;  Lineasli - ]][Ap++] =    ;
                Lee_linea(Linea + 3, 21*j-7, 20, Lineasli - 1], &Ap);
                Lineasli - ]][Ap++] =  , ;  Lineasli - ]][Ap++] =    ;
                } /*Fin for j*/
              Ap -= 2;
              Lineasli - ]][Ap++] =  ) ;  Lineasli - ]][Ap] =  \0 ;
      }  /*Fin switch*/
    Linea = Linea + Diferencia;
    }  /*Fin for i*/
  } /*Fin Lee_esque*/
/**/


/* Leera la cadena de caracteres que aparesca en una de-  */
/* terminada cadena de una determinada linea.             */
/*********************************************************/
/*                                                       */
/*                 PROCEDIMIENTO LEE_LINEA               */
/*                                                       */
/*********************************************************/

Lee_linea(Linea, Columna, Ancho, Cadena, Apuntador)

int Linea,       /* Num. linea de pantalla de la que leera */
    Columna,     /* Columna de pantalla apartir de la que leera */
    Ancho,       /* de la columna del esqueleto de la que leera */
```

```
    *Apuntador; /* Indica sig. caract. de la Cadena en el que es-
                /* cribira el caracter recien leido de pantalla */
char *Cadena;   /* String en que escribira la forma lineal de pregunt. *

{
int i, Frente, Fondo, cont;
char Caracter;
char scread();

for(i = (Columna + Ancho -1); i > Columna; i--)
  {
  scursset(Linea, i);
  Caracter = scread(&Frente, &Fondo);
  if (Caracter != ' ') break;
  } /*Fin for */
cont = Columna;
while (cont <= i)
  {
  scursset(Linea, cont);
  Caracter = scread(&Frente, &Fondo);
  Cadena[(*Apuntador)++] = Caracter;
  cont++;
  } /*Fin while*/
} /*Fin Lee_linea*/
```

```
anta(Rel_1, Rel_2, Rel_3, Rel_4)

char *Rel_1;
char *Rel_2;
char *Rel_3;
char *Rel_4;

  {
  Rel_1[0] = Rel_2[0] = Rel_3[0] = Rel_4[0] =    ;
  Rel_1[1] = Rel_2[1] = Rel_3[1] = Rel_4[1] = '\0';
  Clear_scr();
  Dib_ventana(5, 74, 6);
  Menu_1(Rel_1, Rel_2, Rel_3, Rel_4);
  }  /*Fin Pantalla*/


/* Limpia la Pantalla y activa la primer pagina de esta   */
/*********************************************************/
/*                                                       */
/*            PROCEDIMIENTO CLEAR_SCR                    */
/*                                                       */
/*********************************************************/

Clear_scr()
  {
  scclear();
  scpage(0);
  } /*Fin Clear_scr*/
/**/


/* Dibuja la ventaja en la que se presentaran las opciones*/
/* para seleccionar los tipos de esqueletos que se usaran */
/* para la solicitud                                      */
/*********************************************************/
/*                                                       */
/*            PROCEDIMIENTO DIB_VENTANA                  */
/*                                                       */
/*********************************************************/

Dib_ventana(izq, der, lin)

  int izq, der, lin;
  {
  int cont;

  scCursor(1, izq);
  scWrite((char)0xC9, 1);
  scCursor(1, lin );
  scWrite((char)0xCD, 68);
  scCursor(1, der );
  scWrite((char)0xBB, 1);
  for(cont = 2; cont <= 21; cont++)
    {
```

```
    scCurset(cont, izq);
    scWrite((char)0xba, 1);
    scCurset(cont, der);
    scWrite((char)0xba, 1);
    } /*Fin for cont*/
  scCurset(22, izq);
  scWrite((char)0xc8, 1);
  scCurset(22, lin );
  scWrite((char)0xcd, ee);
  scCurset(22, der);
  scWrite((char)0xbc, 1);
  } /*Fin Dib_ventana*/
/**/


/* Presenta el menu que estara en la ventana que servira  */
/* para seleccionar los tipos de esquemas.                */
/*********************************************************/
/*                                                       */
/*            PROCEDIMIENTO MENU_1                       */
/*                                                       */
/*********************************************************/

Menu_1(Rel_1, Rel_2, Rel_3, Rel_4)

char *Rel_1, *Rel_2, *Rel_3, *Rel_4;

  {
  char *mensaje1, *mensaje2, mensaje[11], seleccion, Relacion[9];
  int cont = 1;

  seleccion =    ;
  mensaje[10] = '\0' ;  mensaje[1] =  ) ;  mensaje[2] =    ;
  mensaje1 = "** Sistema de Consulta Grafico de Bases de Datos **";
  mensaje2 = "Deme el nombre de la Relacion que le corresponde: ";
  scdspmsg( 3, 14, 7, 0, mensaje1);
  scdspmsg( 4, 32, 7, 0, "Tipo");   scdspmsg( 4, 38, 7, 0, "=");
  scdspmsg( 4, 40, 7, 8, "Q B E");  scdspmsg( 4, 46, 7, 0, "=");
  scdspmsg( 8,  8, 7, 0, "-- Seleccione  el ler. tipo  de");
  scdspmsg( 9, 10, 7, 8, "Esqueleto");
  scdspmsg( 9, 21, 7, 0, "que usara  en  su");
  scdspmsg(10, 10, 7, 0, "solicitud:  ");
  scdspmsg( 7, 47, 7, 0, "-Relacion .............1");
  scdspmsg( 8, 47, 7, 0, "-Caja de Condicion ....2");
  scdspmsg(11, 47, 7, 0, "-Tabla de Resultados ..3");
  scdspmsg(21, 43, 7, 8, "Presione 'ESC  para continuar");
  while (seleccion != (char)0x1b)
    {
    scCurset(10, 22);
    seleccion = getch();
    if (cont > 4)
      break;
    if (seleccion !=  1  && seleccion !=  2  && seleccion !=  3  &&
        seleccion != (char)0x1b)
```

```
      {
      scdspmsg(2), 10, 7, 8, "Seleccion incorrecta");
      continue;
      } /*Fin if seleccion*/
    scdspmsg(2), 10, 7, 0, "                          ");
    scdspmsg(14, 61, 7, 0, "          ");
    if (seleccion == (char)0x1b) break;
    strcpy(Relacion, "         ");
    strcpy(&mensaje[3],  "       ");
    switch(seleccion)
      {
      case '1' : scdspmsg(14, 11, 7, 0, mensaje);
                 Lee_rela(Relacion);
                 strcpy(&mensaje[3], Relacion);
                 break;
      case '2': mensaje[3] = mensaje[4] = Relacion[0] = Relacion[1] = 'L' ;
                mensaje[5] = Relacion[2] = '\0' ;
                break;
      case '3': mensaje[3] = 'I' ;  mensaje[4] = 'R';
                Relacion[0] = mensaje[3];  Relacion[1] = mensaje[4];
                mensaje[5] = Relacion[2] = '\0' ;
      } /*Fin switch seleccion*/
    switch(cont)
      {
      case 1: mensaje[0] = (char)0x31;
              strcpy(Rel_1, Relacion);
              break;
      case 2: mensaje[0] = (char)0x32;
              strcpy(Rel_2, Relacion);
              break;
      case 3: mensaje[0] = (char)0x33;
              strcpy(Rel_3, Relacion);
              break;
      case 4: mensaje[0] = (char)0x34;
              strcpy(Rel_4, Relacion);
      } /*Fin switch cont*/
    scdspmsg(14+cont, 10, 7, 0, mensaje);
    cont = cont + 1;
    } /*Fin while*/
  } /*Fin Menu_1*/


/* Lee el nombre de la Relacion para la cual el ususario  */
/* desea se tenga un esqueleto que usara para su solicitud*/
/*************************************************************/
/*                                                           */
/*            PROCEDIMIENTO LEE_RELA                         */
/*                                                           */
/*************************************************************/

Lee_Rela(Relacion)

char *Relacion;
```

```
t i = 61;
t j;

curset(14, i);
r(j = 0; (Re)acion[j] = getch()) != (015); j++)
{
scwrite(Relacion[j], 1);
sccurset(14, i + j + 1);
}
tacion[j] = \0 ;
*Fin Lee_Rela*/
```

El jurado designado por la Sección de Computación del Departamento de Ingeniería Eléctrica del Centro de Investigaciones y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó esta tesis el 6 de Octubre de 1987.

Dr. Guillermo Morales Luna

M. en C. Henrique Contreras G.

Dr. Renato Barrera Rivera