CASOS PRACTICOS EN EL DISEÑO DE SISTEMAS DE INFORMACION 1980

Fecha	Tema	Hora	Profesor
Agosto 8	Banco de Datos DETENAL	18 a 21 h	Ing. Ernesto Bribiesca Correa
" 9	Banco de Datos Niño Mexicano	9 a 14 h	Ing. Rafael Domínguez
" 15	Sistema OCI para Control de Ventas en una Cadena de distribuidoras	18 a 21 h	Dr. Adolfo Guzmán Arenas
" 16	Temas Especiales en bases de Datos Tendencias recientes en bases de datos Cambios y mejoras.(El Problema de querer demasiados cambios demasiado pronto) Temas Especiales en Bases de Datos Bases de datos distribuidos como una alternativa del futuro.	9 a 11 a. m. 11 a 13 h 13 a 14 h	Dr. Alejandro Buchmann Dr. Adolfo Guzmán Arenas. Dr. Alejandro Buchmann
" 22	Objetivos: Claridad. Especificaciones escritas. Ejemplo de especificaciones de diseño. Lenguajes: Ventajas y desventajas de los lenguajes más comunes.	18 a 20 h 20 a 21 h	Dr. Adolfo Guzmán Arenas
" 23	Características importantes de un sistema de información y como implementarlas. Habilidad para responder a preguntas generales Habilidad para desplegar los resultados en cualquier formato. Formas de guardar la información: * Por items * Por posición (propiedades implícitas) * Propiedades explícitas * Ordenadas * Desordenas * Por vectores de acceso (listas invertidas) * Almacenamiento híbrido	9 a 14 h	Dr. Adolfo Guzmán Arenas

Fecha	Tema	Hora	Profesor
Agosto 29	Formas de consultas de información * Propiedades reales * Propiedades virtuales * Propiedades computadas o virtuales * Propiedades de conjunto * Propiedades codificadas (criptografía privacidad)	18 a 21 h	Dr. Adolfo Guzmán Arenas
	Manejo de textos en los sistemas de información		
	Totalizadores, histogramas y diagramas de dispersión		
Agosto 30	Habilidad para invocar funciones (subrutinas) arbitrarias	9 a 12 h	Dr. Adolfo Guzmán Arenas
	Habilidad para ser llamado desde cualquier programa		
	Lector General. Habilidad para poder leer un archivo con formato arbitrario.		
	Escritor General. Habilidad para poder generar un archivo con un formato arbitrario.		
	Bases de datos reconfigurables	12 a 14 h	M. en C. Víctor Germán Sánchez.
Sept. 5	Organización del grupo de trabajo para el diseño y programación.	18 a 19 h	M. en C. Marcial Portilla Robertson.
	Alimentación de la información. El problema de mantenimiento actualizado el sistema.	19 a 20 h	" " "
	Documentación	20 a 21 h	" " "

Fecha	Tema	Hora	Profesor
Sept. 6	Entrenamiento al usuario	9 a 10 a.m.	Ing. Rafael Domínguez de León
	Compra/alquiler vs hacer un sistema de información	10 a 12 h	Dr. Adolfo Guzmán Arenas
	Discusión y Conclusiones.	12 a 14 h	" " " "

Directorio de Profesores del Curso Casos Prácticos en el Diseño de Sistemas de Información 1980.

1. Ing. Ernesto Bribiesca Correa
Asesor
Dirección General de Geografía del Territorio Nacional
Secretaría de Programación y Presupuesto
S. A. Abad No. 124- 1° Piso
México 8, D.F.
5 78 62 00 Ext. 176

2. Dr. Alejandro Buchmann Sauter
Investigador
IIMAS
UNAM
5 50 5 2 15 Ext. 4577

3. Ing. Rafael Domínguez de León
Director General
Informática, Ingeniería y Administración, S.A.
Alpes No. 24-303
México 20, D.F.
651 95 85

4. M. en C. Marcial Portilla Robertson
Jefe de la División de Computación
Facultad de Ingeniería
UNAM
México 20, D.F.
550 52 15 Ext. 3746

5. Dr. Adolfo Guzmán Arenas
Jefe del Proyecto Arquitectura Heterárquicas Reconfigurables
Instituto de Investigaciones Matemáticas Aplicadas a Sistemas
UNAM
México 20, D.F.
550 52 15 Ext. 4585

6. M. en C. Víctor Germán Sánchez Arías
Departamento de Sofstware
Centro de Servicios de Cómputo
UNAM
550 52 15 Ext. 4537

'edcs.

centro de educación continua
división de estudios de posgrado
facultad de ingeniería unam

CASOS PRACTICOS DEL DISEÑO DE SISTEMAS DE INFORMACION

MANUAL DE USUARIO PARA LA EXPLOTACION DE UN BANCO DE DATOS GEOGRAFICOS

ERNESTO BRIBIESCA CORREA
ADOLFO GUZMAN ARENAS

AGOSTO, 1980

CENTRO CIENTIFICO DE AMERICA LATINA

Reporte CCAL-74-17

Diciembre 1974

MANUAL DE USUARIO PARA LA EXPLOTACION DE UN BANCO DE DATOS GEOGRAFICOS

Ernesto Bribiesca Correa

Adolfo Guzmán Arenas

Centro Científico de América Latina
IBM de México, S. A.
Mariano Escobedo 595
México 5, D. F.

MANUAL DE USUARIO

PARA LA EXPLOTACION DE UN BANCO DE DATOS GEOGRAFICOS

C O N T E N I D O

-RESUMEN.....1

-INTRODUCCION.....2

-PROPOSITO.....3

-OBJETIVO.....4

-DEFINICION DE PAIS, ZONA, CUADRO Y SUBCUADRO.....5

-FORMA GRAFICA DE SALIDA.....9

-EJEMPLOS INTRODUCTORIOS.....12

-DESCRIPCION DE LAS FUNCIONES RELACIONALES.....16

-DESCRIPCION DE LAS FUNCIONES LOGICAS.....20

-FUNCIONES NUMERICAS.....45

-LAS FUNCIONES EVALUA, EVALU Y EVAL.....46

-LA FUNCION CERCA.....58

-EJEMPLOS USANDO COMBINACIONES DE TODAS LAS FUNCIONES.....64

-APENDICE.....87

-AGRADECIMIENTOS.....112

-REFERENCIAS.....113

-REPORTES DEL CENTRO CIENTIFICO IBM DE AMERICA LATINA PARA 1974.....114

RESUMEN

Este manual describe un lenguaje de acceso a datos geográficos a través de programas escritos en el lenguaje FORTRAN IV.

Los datos geográficos, son los que normalmente se encuentran en los mapas de CETENAL.

Una característica de este lenguaje es que dentro de las restricciones que otorga FORTRAN IV es ilimitado en su construcción de predicados ó preguntas. También tiene la facilidad para el usuario de un rápido aprendizaje, sin tener conocimientos previos de programación ó de computación.

El objetivo de este trabajo es poder contestar preguntas en forma rápida, las que manualmente serían muy laboriosas y emplearían mucho tiempo. Como por ejemplo, en qué lugares puede crecer más maíz en la República Mexicana; qué partes de República Mexicana no están cubiertas a más de 60 Km. por sus redes de caminos y carreteras existentes; qué lugares deben ser evacuados en casos de sismos; qué lugares son apropiados para el turismo; qué lugares tienen más erosión y necesitan control inmediato; en qué lugares hay pueblos; en qué lugares hay escuelas; qué pueblos tienen más de 1,000 habitantes y no tienen medios de comunicación.

Con este Banco de Datos Geográficos ya se han localizado cuencas lecheras en el área de Ojo Caliente, Zac. (referencia 4). Sin embargo, este programa apenas indica el inicio de una serie de aplicaciones en planificación regional, estudios económicos, etc.

INTRODUCCION

La utilización y explotación de un Banco de Datos Geográficos es de gran importancia, ya que nos permite contestar infinidad de preguntas con rapidez y con buen grado de confiabilidad, pues ésto realizado en forma manual es sumamente laborioso.

Con el entendimiento y comprensión de este manual, el usuario puede crear todos los predicados posibles, sin tener conocimientos previos de computación ó de programación, pero este manual no incluye en forma alguna la lógica de los programas.

CETENAL (Comisión de Estudios del Territorio Nacional) es una dependencia de la Secretaría de la Presidencia, que se fundó hace 6 años, la cual elabora mapas con mucha precisión é información por medio de aerofotogrametría de toda la superficie de la República Mexicana que se dividió en zonas, una zona tiene una superficie de 1,000 Km^2 y cada zona tiene 5 cartas geográficas que son:

- Carta geológica
- Carta topográfica
- Carta uso del suelo
- Carta edafológica
- Carta uso potencial

Cada zona utiliza de 80 a 100 fotografías con un traslape del 60%. La escala de las cartas geográficas es variada, mas nuestro banco por el momento está usando cartas de escala 1:50 000.

Además la CETENAL elabora otras cartas: Carta Urbana, de climas, etc., que no están incluidas en el presente estudio. La Comisión está realizando el levantamiento aerofotográfico de nuestro país, técnica que le sirve para llevar a cabo la moderna Cartografía de México, mediante la aplicación de métodos avanzados. El estudio cartográfico tiene por objeto investigar y ubicar los recursos naturales del país; conocer el uso que tiene en la actualidad el territorio; precisar el equipamiento urbano de todas las localidades, así como las obras de infraestructura. En síntesis: se trata del Inventario Nacional.

Las fotografías se toman en un avión que vuela a una altura de - 25,000 a 50,000 pies. Estas fotografías se toman periódicamente con un ciclo de 10 años. El total de area fotografiada de la República Mexicana es del 70% aunque de ésta área no existen todos los mapas ya elaborados pero sí un gran porcentaje.

PROPOSITO.

Poder crear en forma automática y sistemática un conjunto determinado de preguntas para la explotación de un Banco de Datos Geográficos por medio de funciones lógicas sin que el usuario tenga conocimientos previos de computación o programación.

OBJETIVO

Contestar preguntas en forma rápida, las que manualmente serían - muy laboriosas y emplearían mucho tiempo. Como por ejemplo, en qué lu-- gares puede crecer más maíz en la República Mexicana, qué partes de la- República Mexicana no están cubiertas a más de 60 Km. por sus redes de- caminos y carreteras existentes, qué lugares deben ser evacuados en casos de sismos, qué lugares son apropiados para el turismo, qué lugares son- apropiados para zonas industriales, qué lugares tiene más erosión y nece sitan control inmediato, en qué lugares hay escuelas, qué pueblos tienen más de 1,000 habitantes y no tienenmedios de comunicación, qué lugares son buenos para la ubicación de depósitos de agua, en qué porcentaje se presenta el pastizal inducido en determinada zona; qué pueblos no es tán a más de 10 Km de una aeropista, qué pueblos tienenabastecimiento de agua por medio de pozos; itiene determinado pueblo carretera pavimen- tada con tal pueblo? ¿cuáles son los servicios propuestos para el pueblo de san Nicolás? cuales son los lugares buenos para pasar carreteras, lí neas eléctricas, telefónicas, telegráficas, etc.

Con este Banco de Datos Geográficos ya se han localizado cuencas lecheras en el área de Ojo Caliente, Zac.(referencia 4.). Sin embargo este programa apenas indica el inicio de una serie de aplicaciones en planificación regional, estudios económicos, etc.

DEFINICION DE PAIS, ZONA, CUADRO Y SUBCUADRO.

La República Mexicana está dividida en aproximadamente 2,336 zonas; cada zona cuenta con área aproximada de 1,000Km^2 y se divide en 48 cuadros que están numerados del 1 al 48, cada cuadro tiene un área de 25 - Km^2 y a su vez cada cuadro se divide en cuatro subcuadros de una longitud de 2.5Km y un área total de 6.25 Km^2 ; los subcuadros están definidos por a,b,c,d y por el cuadro al que pertenecen, los cuadros se definen por la zona a la que pertenecen.

En la figura 1 vemos una descripción gráfica de las propiedades - de un cuadro y de un subcuadro.

Tanto el país como la zona, el cuadro y el subcuadro están definidos por números de nivel que son:

SALIDA	NIVEL
País	1
Zona	2
Cuadro	3
Subcuadro	4

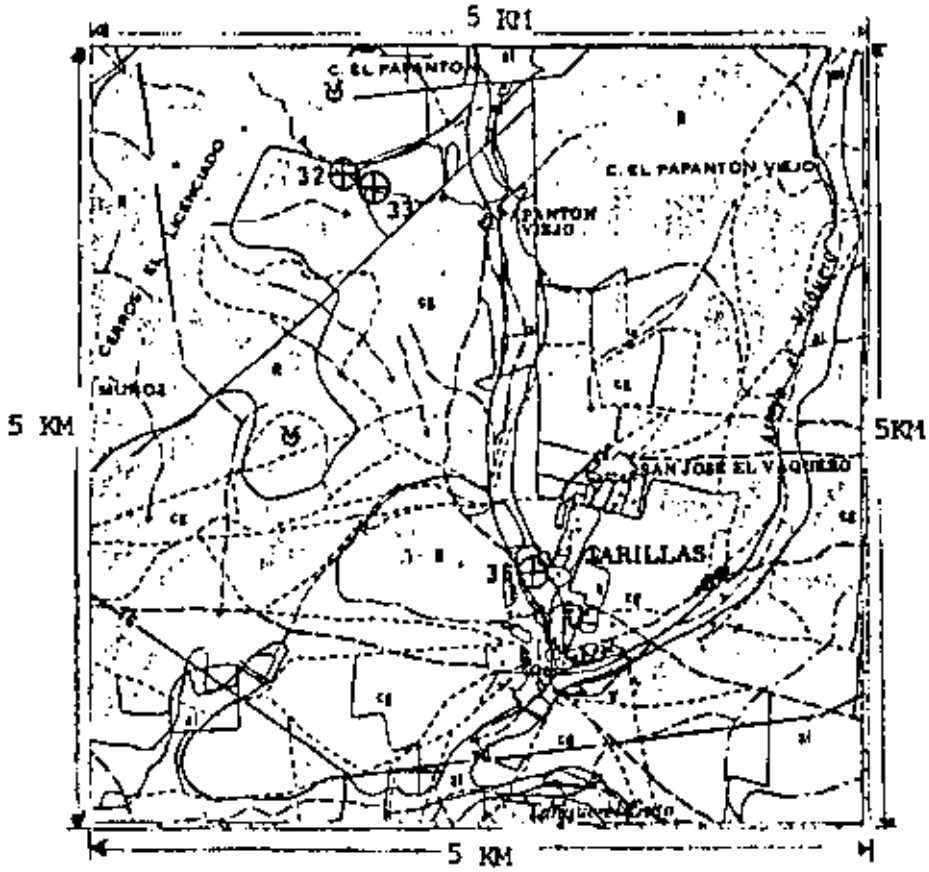
O sea nosotros podemos buscar propiedades al nivel que se desee.*

En la figura 2 observamos los niveles propuestos para la República Mexicana.

Un país (nivel 1) tiene 2,336 zonas (nivel 2) la división de México en zonas ha sido hecha por Cetenal, según la carta de Cetenal, la que - así mismo nos explica la nomenclatura seguida, para nuestro estudio hemos escogido la carta F-13 B69 denominada "Ojo Caliente", del estado de Zacatecas.

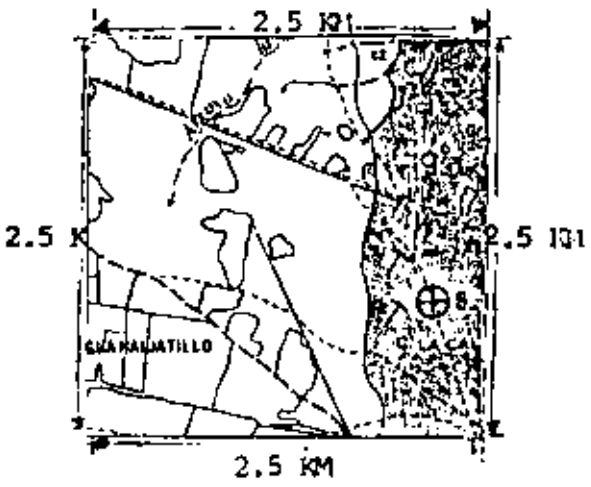
*En la versión que éste reporte describe, solo se han implementado los niveles 3 y 4. En una versión posterior que tal vez se lleve a cabo con la colaboración de CETENAL, más niveles serán implementados.

DEFINICION DE CUADRO Y SUBCUADRO



CUADRO

AREA TOTAL DE CUADRO = 25 KM^2



SUBCUADRO

AREA TOTAL DE SUBCUADRO = 6.25 KM^2

PAIS	NIVEL 1
ZONA	NIVEL 2
CUADRO	NIVEL 3
SUBCUADRO	NIVEL 4

FIG.1 RELACIONES ENTRE CUADRO Y SUBCUADRO

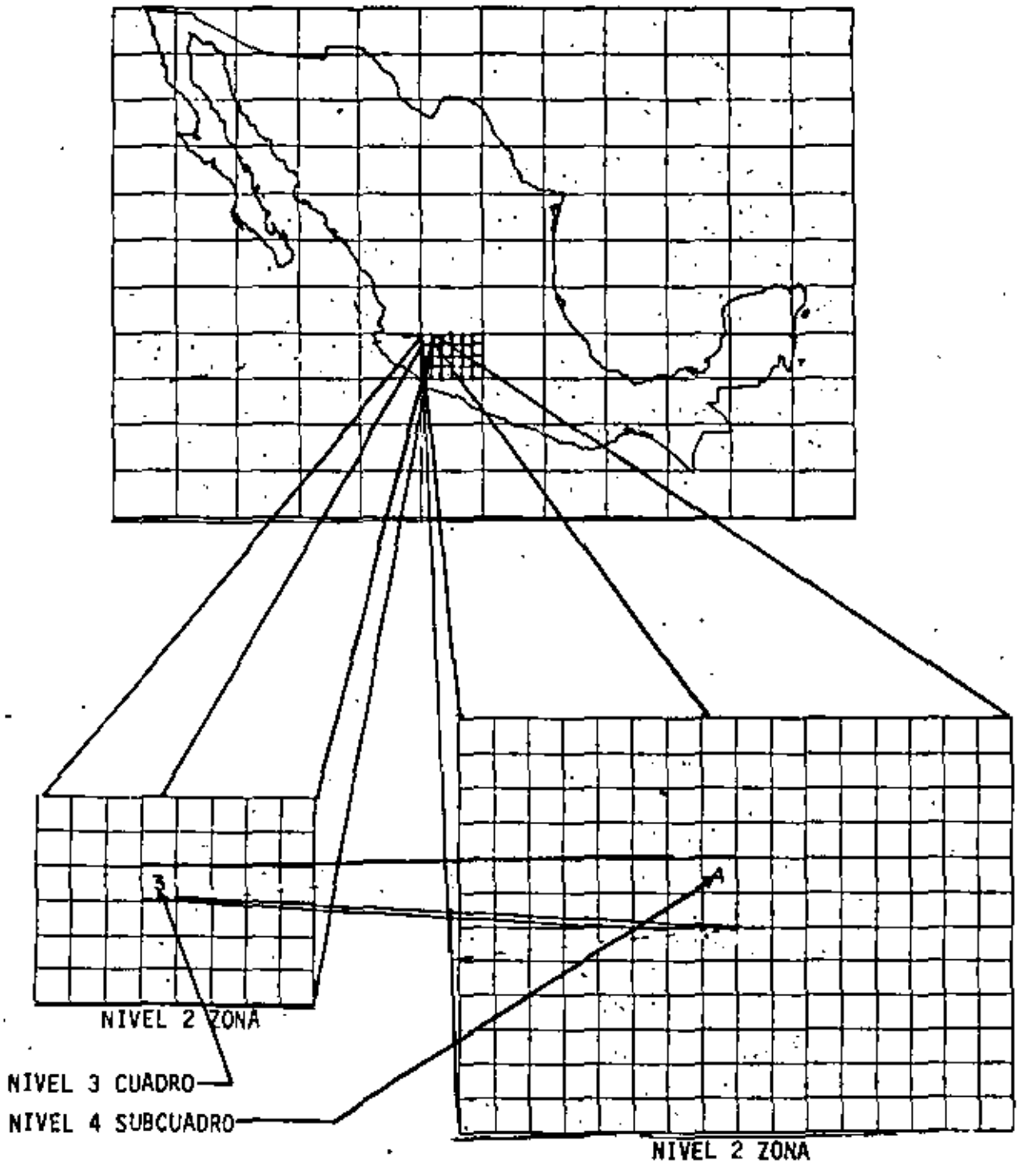


FIG. 2 NIVELES DE LA REPUBLICA MEXICANA.

Una zona (nivel 2) tiene 48 cuadros (nivel 3), denominadas cuadro (1, 1), cuadro (1, 2), ..., cuadro (6, 8); la división de una carta en cuadros está hecha por los paralelos y meridianos que la carta tiene impresos, los paralelos cada 2.5 minutos (5Km aprox.) y los meridianos cada 2.5 minutos (también 5 Km).

FORMA GRAFICA DE SALIDA.

Como respuesta a nuestra búsqueda a un nivel determinado, la máquina produce en la impresora un arreglo de ceros y unos, que expresa la función característica del predicado usado (ver figura 3).

Por ejemplo, si buscamos a nivel de cuadros (nivel 3) los lugares (claro,cuadro en este caso) en donde cierto predicado ("FRIJOL", digamos) es cierto ("es cierto" significa que "se satisface en ese lugar", o sea que "en ese lugar las condiciones buscadas (y definidas en el predicado "FRIJOL") se han encontrado," o sea que "en ese lugar el predicado adquiere el valor'cierto' o 'T' o '1' "), en cierto momento nosotros decimos

CALL BUSCA (FRIJOL, 3)

y como respuesta o resultado obtenemos el arreglo de dimensiones 6 x 8

0	1	0	0	0	0	0	0
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Que con 1's nos marca los lugares a nivel 3 (los cuadros) donde hay "FRIJOL", es decir, donde el predicado "FRIJOL" se satisface o es cierto. Los ceros nos indican que en esos cuadros no hubo "FRIJOL", es decir, que el predicado buscado no se satisfizo en ese lugar, o lo que es lo mismo, que el conjunto de propiedades definido (ya después veremos cómo) por el predicado "FRIJOL" no se encuentra (no existe) en los lugares donde aparece un 0. Resumiendo, "1" significa "sí hay acá lo que buscabas" y "0" significa "no hay acá lo que buscabas".

En el ejemplo anterior, los cinco 1's nos dicen los lugares donde existe "FRIJOL" son los cuadros 2, 13, 14, 21 y 22 de la zona interrogada.

Si la busqueda se hace a nivel 4, nivel de subcuadro, aparecerá una matriz de dimensiones 16X12 que nuevamente nos informa con 1's

de los lugares donde "FRIJOL" existe. Por ejemplo, el resultado podía ser

```
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Hemos marcado con líneas de puntos los cuatro subcuadros $\begin{matrix} a & b \\ c & d \end{matrix}$ que se refiere al cuadro 2, y también los que se refiere al cuadro 13 anteriormente, cuando hicimos la búsqueda a nivel 3, supimos que los cuadros 2, 13 (entre otros) tenían "FRIJOL". Ahora sabemos algo más; que los cuatro subcuadros del cuadro 2 tienen "FRIJOL", en tanto que sólo los subcuadros b y c del cuadro 13 tienen "FRIJOL".

RESULTADOS DE COMPUTADORA POR MEDIO DE ARREGLOS NUMERICOS

```
 1   2   3   4   5   6   7   8
 9  10  11  12  13  14  15  16
17  18  19  20  21  22  23  24
25  26  27  28  29  30  31  32
33  34  35  36  37  38  39  40
41  42  43  44  45  46  47  48
```

CUADROS

```
a  b  a  b  a  b  a  b  a  b  a  b  a  b  a  b
c  d  c  d  c  d  c  d  c  d  c  d  c  d  c  d
a  b  a  b  a  b  a  b  a  b  a  b  a  b  a  b
c  d  c  d  c  d  c  d  c  d  c  d  c  d  c  d
a  b  a  b  a  b  a  b  a  b  a  b  a  b  a  b
c  d  c  d  c  d  c  d  c  d  c  d  c  d  c  d
a  b  a  b  a  b  a  b  a  b  a  b  a  b  a  b
c  d  c  d  c  d  c  d  c  d  c  d  c  d  c  d
a  b  a  b  a  b  a  b  a  b  a  b  a  b  a  b
c  d  c  d  c  d  c  d  c  d  c  d  c  d  c  d
a  b  a  b  a  b  a  b  a  b  a  b  a  b  a  b
c  d  c  d  c  d  c  d  c  d  c  d  c  d  c  d
```

SUBCUADROS

FIG. 3 RESULTADOS DE COMPUTADORA POR IMPRESORA.

EJEMPLOS INTRODUCTORIOS

En el resto de este reporte, estaremos usando las cinco cartas (edafológica, topográfica, de uso actual del suelo, de uso potencial del suelo y geológica), que se refieren a la zona F 13 B 69, de Ojo Caliente, Zacatecas. La información que de ellas se encuentra en nuestro banco de datos geográficos se encuentra descrita en las tablas localizadas al final de este reporte y corresponden, aproximadamente, a la información que una persona "promedio" obtendría de esas cartas, haciendo uso de las leyendas y explicaciones que aparecen en los márgenes de éllas.

Las funciones que accesan el banco de datos nos permiten descubrir los sitios (cuadros ó subcuadros) que poseen determinada propiedad.

Procedemos a dar algunos ejemplos sencillos, con el fin de mostrar la simplicidad de su uso. Cualquiera de estos puntos obscuros en estos ejemplos serán aclarados en secciones posteriores.

Por ejemplo, si deseamos descubrir los cuadros (nivel 3) que poseen más de un 20% de cultivos, o sea aquéllos cuadros que tengan más de 20% de superficie cultivada, definimos el predicado CULTI (el nombre no importa, mas debe tener de 1 a 6 letras; es conveniente usar nombres mnemónicos) mediante el postulado que nos dice que hemos definido "CULTI" como todo aquél lugar donde la propiedad 100 (que en la tabla 1 vemos que nos indica CULTIVOS) sea mayor de 20 (ó sea mayor de 20%).

Una vez hecha esta definición, procedemos a buscar "CULTI" a nivel de cuadro (nivel 3) como sigue:

```
CULTI = PRO (100.,MAYORQ,20)
CALL BUSCA (CULTI,3)
```

Los tres postulados anteriores necesitan de ciertos "accesorios" en forma de postulados adicionales; los programas completos son:

```
LOGICAL FUNCTION CULTI(N)
LOGICAL PRO, MAYORQ
EXTERNAL MAYORQ                    ← Programa que define el
CULTI = PRO (100., MAYORQ, 20)       el predicado "CULTI"
RETURN
END
```

```
LOGICAL CULTI
EXTERNAL CULTI
CALL BUSCA(CULTI,3)
STOP
END
```

Programa principal que busca → "CULTI" a nivel 3 (de cuadro)

y el resultado es un arreglo de ceros y unos representándonos los cuadros (de la zona F 13 9 69) que carecen (ceros) ó tienen (unos) la propiedad CULTI, ó sea que tienen más del 20% de su superficie cubierta por cultivos:

```
0 0 0 0 1 1 1 0
1 1 0 0 0 1 1 0
1 1 1 0 0 0 0 0
1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
```

NOTAS:

Brevemente, las declaraciones LOGICAL en los programas definen a CULTI, MAYORQ y PRO como funciones que adquieren valores lógicos (cierto ó falso); la (N) de CULTI (N) es una variable muda; las declaraciones EXTERNAL definen a CULTI y MAYORQ como nombres de funciones y la secuencia RETURN END en el predicado y STOP END en el programa principal terminan "correctamente" los programas. El usuario no necesita entender todo esto, pudiendo mirar a todas estas declaraciones como parafernalia engorrosa pero necesaria: no puede omitirse.

Para buscar la misma propiedad a nivel de subcuadro, diremos en el programa principal:

CALL BUSCA (CULTI,4).

Si estamos interesados por zonas donde más del 60% sean cultivos (propiedad 100.) ó pastizales (propiedad 102. según la tabla 1) diremos:

ZONA = PRO (100., MAYORQ, 60) .OR. PRO (102., MAYORQ, 60)

Si queremos hallar zonas que no tengan chaparrales (propiedad 103.) diremos:

NOCHAP = .NOT. PRO (103., MAYORQ,0)

Es decir, PRO(103., MAYORQ, 0) define lugares donde sí existen chaparrales, y con un .NOT. enfrente se niega esta propiedad.

¿Qué lugares tienen pocos álamos y muchos pirules? Si convenimos en que "poco" signifique menos del 15% y "muchos" signifique más del 70%, nuestra función es:

ALAPIR = PRO (238., MENORQ, 15) .AND. PRO (245., MAYORQ, 70)

¿Qué lugares están comunicados ya bien sea por carretera ó ferrocarril? Si vemos la tabla 1, observaremos que las propiedades de la 110 a la 116 se refieren a diferentes tipos de carreteras, en tanto que las propiedades 117 a 120 se refieren a tipos de vías de ferrocarril. Una de ellas es suficiente para comunicar el lugar por donde pasan. La función es:

COMUN = UNADE (110, 120, MAYORQ, 0)

que se interpreta diciendo "al menos una de las propiedades 110. a 120. debe ser mayor que cero", es decir, debe existir.

Si quiero conocer todos los lugares que están comunicados, y que tienen agua almacenada (presa, bordo ó depósito), diremos:

COMAGU = UNADE (110, 120, MAYORQ, 0) .AND. UNADE (126,128, MAYORQ, 0)

¿En qué lugares hay palmares (digamos, 25% de la superficie) a más de mil metros sobre el nivel del mar?

```
PALMAR = PRO (104.,MAYORQ,25) .AND. PRO (195.,MAYORQ,1000)
```

¿Qué zonas se encuentran entre 1600 y 1800 metros sobre el nivel del mar?

```
RESUL = PROP (190., ENTRE,1600,1800)
```

¿Crecerán los nopales sobre suelos de tipo Gleysol plíntico?
Definamos "crecen los nopales" cuando más del 30% del suelo está cubierto de nopalera (propiedad 266.) y definamos que un suelo es de tipo "Gleysol plíntico" cuando tiene más del 70% de Gleysol plíntico dominante (propiedad 540.). Entonces,

```
NOPG = PRO (266.,MAYORQ,30) .AND. PRO (540.,MAYORQ,70)
```

Se pueden hacer preguntas arbitrariamente largas. Por ejemplo, un lugar turístico puede ser aquél donde haya playa, (188.), bosque (105. ó 106.) ó ríos (181.) en lugares no muy altos (< 2500 mts.); ó bien lugares altos, pero con volcanes (1160.) ó suelos glaciares (957.); ó bien varios manantiales termales (más de 4) (1163.) pero bien comunicados (una de 110. a 120.) en terrenos no montañosos (que no sea 804. ni 805.)

Podemos proceder por partes:

```
UNO= PRO (188.,MAYORQ,0) .OR.UNADE (105.,106.,MAYORQ,0)
     .OR. PRO (181.,MAYORQ,0)  .AND. PRO (194.,MENORQ,2500)
DOS = PRO (195.,MAYORQ,2500) .AND. (PRO (1160.,MAYORQ,0)
     .OR.PRO (957.MAYORQ,0))
TRES = PRO (1163.,MAYORQ,4) .AND. UNADE (110., 120.,MAYORQ,0)
      .AND. .NOT. UNADE (804.,805.,MAYORQ,0)
```

Y luego decimos finalmente,

```
TURIS = UNO .OR. DOS .OR. TRES
```

NOMENCLATURA USADA

PREDICADO: función lógica definida por el usuario
FUNCIONES RELACIONALES: MENORQ, MAYORQ, DIFERE, IGUALQ y ENTRE
FUNCIONES LOGICAS: PRO, PROP, UNADE, CERCA, SERPRO, SERPOB y PUEBLO

DESCRIPCION DE LAS FUNCIONES RELACIONALES

Las funciones relacionales se usan dentro de las funciones lógicas PRO, PROP y UNADE, para delimitar los valores de una propiedad.

Son útiles para construir el predicado que finalmente utilizamos, ("TRIGO", "TURIS", etc.), es decir, el que contiene todas las propiedades que nos interesan.

REGLA 1. Cuando en la definición de estos predicados se usan alguna de PRO, PROP, UNADE, MAYORQ, MENORQ, ENTRE, DIFERE, IGUALQ, tal función a usarse debe declararse antes de su uso, que es una función lógica. Ejemplo:

Supongamos que quiero hallar los lugares donde se efectúa agricultura de temporal nómada, independientemente del tipo de cultivo (el que a su vez puede ser anual, permanente ó semi-permanente). Esto es, cualquiera de las propiedades 202., 205., ó 208.

Antes de decir,

```
NOMADA = PRO (202., MAYORQ, 0) .OR. PRO (205., MAYORQ, 0)
.OR. PRO (208., MAYORQ, 0)
```

Debo declarar

```
LOGICAL PRO, MAYORQ.
```

REGLA 2. Cuando en la definición de estos predicados se usan alguna de MAYORQ, MENORQ, ENTRE, DIFERE, IGUALQ, debe declararse antes de su uso, que es una función externa, mediante la declaración EXTERNAL, la que puede ir después ó antes de la declaración LOGICAL.

De modo que el ejemplo anterior nos puede quedar ya completo:

```
LOGICAL FUNCTION NOMADA (N)
LOGICAL PRO, MAYORQ
EXTERNAL MAYORQ
NOMADA = PRO (202., MAYORQ, 0) .OR. PRO (205., MAYORQ, 0) .OR. PRO
(208., MAYORQ, 0)
RETURN
END
```

REGLA 3. Todo predicado final (NOMADA, en este caso), debe declararse: LOGICAL (con LOGICAL FUNCTION...) al momento de definirlo, tal como lo hemos hecho. También debe de declararse LOGICAL y EXTERNAL, en el momento de usarse, es decir, en el programa principal:

```
LOGICAL NOMADA
EXTERNAL NOMADA                    Programa principal que busca
CALL BUSCA (NOMADA, 4)             agricultura nómada a nivel de
STOP                               subcuadro.
END
```

Nótese que el programa principal es "sencillo" y que todas las funciones PRO, MENORQ, etc., se usan en el predicado por el usuario, más no en el programa principal, el que sólo usa BUSCA.

MAYORQ

La función MAYORQ es cierta en una zona, si tal zona posee la propiedad PROP en una cantidad mayor que n. Ahora bien, puesto que casi todas las propiedades están expresadas en porcentajes, es decir, en números enteros del 0 al 100, con mucha frecuencia n es un porcentaje.
Ejemplo:

PRO(100., MAYORQ, 20) será cierta en los lugares que tengan más de 20% de cultivos.
Ejemplo:

PRO (310., MAYORQ, 500) halla lugares con más de 500 habitantes. Todas las funciones relacionales se usan sólo dentro de las funciones lógicas PROP,PRO y UNADE. PRO se usa cuando se necesitan tres argumentos (como en este caso), y PROP cuando se requieren cuatro. Luego entonces MAYORQ, sólo se usa dentro de PRO ó UNADE.

La función relacional MAYORQ nos sirve para determinar que zonas tienen un porcentaje mayor de una propiedad que se desee ó de una altitud, etc.

MENORQ

La función relacional MENORQ nos sirve para determinar que zonas tienen un porcentaje menor de una propiedad que se desee, de una altitud, etc.

PRO (PROP, MENORQ, n) es cierta en una zona si tal zona posee la propiedad PROP en una cantidad menor que n. PROP = número de propiedad. En punto flotante. n = número. Entero, generalmente del 0 al 100.
Ejemplo:
PRO (277., MENORQ, 30) será cierta en los lugares en que los crasirosulifolios espinosos no lleguen a cubrir el 30% de la superficie de tal lugar.

MENORQ, sólo se usa con PRO ó UNADE.

ENTRE

PROP (PROP, ENTRE, n min, n max) es cierta en los lugares en que el valor de la propiedad PROP está entre n min y n max, es decir, en los que

$$nmin \leq valor \leq nmax$$

PROP es el número de la propiedad (en punto flotante), n min y n max son los límites inferior y superior respectivamente, ambos en punto fijo.
Ejemplo:
PROP (180., ENTRE, 1, 3) nos hallará todos los lugares (cuadros ó subcuadros) donde existan uno, dos ó tres puentes.

La función ENTRE nos sirve para encontrar las zonas que tienen una propiedad que está entre dos valores dados, que puede ser en porcentajes ó en altitudes ó en número de algo.

ENTRE sólo se usa con PROP.

DIFERE

PRO (PROP, DIFERE, n) será cierta en los lugares donde el valor de la propiedad PROP sea diferente de n. PROP es el número en punto flotante de la propiedad buscada y n es un entero.
Ejemplo:
PRO(312., DIFERE, 1) halla los cuadros ó subcuadros (según el nivel a que se le use con BUSCA) que no tienen exactamente un pueblo, es decir, aquéllos que tienen más de uno (dos, tres...pueblos) ó que carecen de pueblo.

La función DIFERE encuentra las zonas que tienen una propiedad diferente a cierta cantidad dada.

DIFERE se usa sólo con PRO ó con UNADE.

IGUALQ

PRO (PROP, IGUALQ, n) es cierta en los lugares en que el valor de la propiedad PROP = n. PROP es el número de la propiedad interrogada, expresado en punto flotante y n es un entero.
Ejemplo:
PRO(190., IGUALQ, 2500) nos halla aquéllos lugares situados a 2500 mts. de altura.

La función relacional IGUALQ encuentra las zonas que tieren una propiedad igual a cierta cantidad dada.

IGUALQ se usa con PRO ó con UNADE.

DESCRIPCION DE BUSCA

BUSCA encuentra la función deseada y a qué nivel.

Ejemplo:

```
CALL BUSCA (FUNCION, NIVEL)
```

DESCRIPCION DE LAS FUNCIONES LOGICAS

PRO (a1, a2, a3)

Es una función lógica que nos sirve para la construcción de predicados ó sea preguntas. Pro es una función que consta de 3 argumentos, donde:

a1 es la propiedad, en punto flotante

a2 es una función relacional y debe ser una de: MAYORQ, MENORQ, DIFERE, IGUALQ.

a3 es la cantidad de relación, en punto fijo.

a1, es la propiedad que se desea y se encuentra su equivalente numérico en la tabla #1 (Ver Apéndice).

Ejemplo:

Usando la función pro y la función relacional MAYORQ y busca. Deseamos saber en qué cuadro hay más de 500 habitantes.

En la tabla #1 (Ver Apéndice), la propiedad que nos da el número de habitantes es la 310.

Definimos nuestro predicado:

```
GENTE = PRO (310., MAYORQ, 500)
```

a) Definimos nuestra LOGICAL FUNCTION con el predicado:

```
      LOGICAL FUNCTON GENTE  (N)
      LOGICAL PRO, MAYORQ
      EXTERNAL MAYORQ
      GENTE = PRO (310., MAYORQ, 500)
      RETURN
      END
```

b) Definimos el programa principal

```
LOGICAL GENTE
EXTERNAL GENTE
CALL BUSCA (GENTE, 3)
STOP
END
```

Como vemos el listado 1, los unos nos indican los cuadros donde hay más de 500 habitantes, los ceros nos indican donde no hay habitantes ó hay menos de 500 habitantes.

Esto se puede verificar en la figura 4, donde como se ve en el cuadro #21 la suma del total de habitantes de este cuadro excede de 500 ya que en el pueblo de Unión de San Antonio hay 480 habitantes y en el pueblo de San Ramón hay 247 habitantes. Aparece un uno.

Para el cuadro #22 como el pueblo Gral. Pánfilo Natera tiene 2,305 habitantes aparece un uno, para el cuadro #29 sólamente hay 205 habitantes que pertenecen al pueblo de Tahonas y vemos que aparece un cero; para el cuadro #30 hay 520 habitantes en el pueblo de Rancho Nuevo y por lo tanto aparece un uno. Esto lo vemos en la figure 4 que pertenece a la carta de uso del suelo.

UNADE (a1, a2, a3, a4)

Es una función lógica que consta de 4 argumentos y es un .OR.

Donde:

a1 es el número de propiedad, en punto fijo

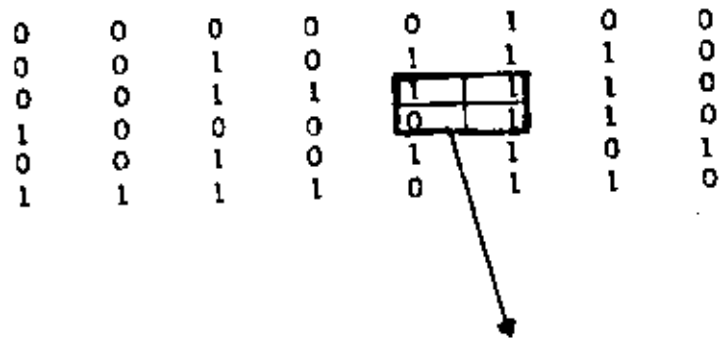
a2 es el número de propiedad

a3 es la función relacional que debe ser una de : MAYORQ, MENORQ, DIFERE, IGUALQ.

a4 Es la cantidad que la función relacional usa para su comparación

```
      LOGICAL FUNCTION GENTE(N)
      LOGICAL MAYORQ,PRO
      EXTERNAL MAYORQ
      GENTE=PRO(310.,MAYORQ,500)
C QUIERO CUADROS QUE TENGAN MAS DE 500 HABITANTES
      RETURN
      END
```

```
      LOGICAL GENTE
      EXTERNAL GENTE
      CALL BUSCA(GENTE,3)
      STOP
      END
```



Equivalente a la figura 4

Listado 1

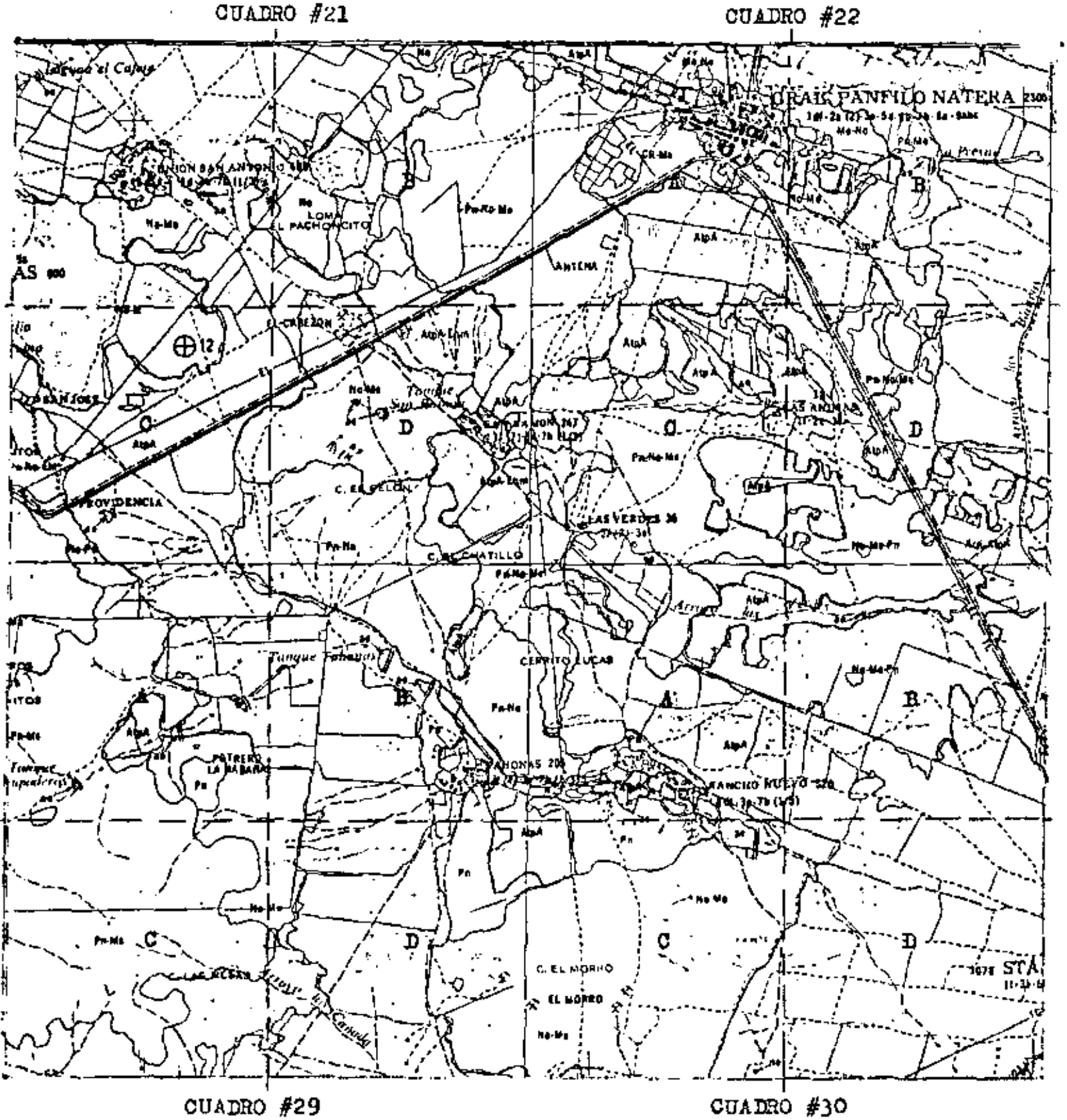


Fig 4 Parte de la Carta de Uso del Suelo de Ojo Caliente, Zac. Nuestro programa ha encontrado aquí sitios con más de 500 habitantes.

Si nosotros deseamos la existencia de una ó varias propiedades consecutivas no importando cual de ellas exista, usamos la función lógica UNADE, si tenemos UNADE (100, 105, MAYORQ, 20) indica que si cualquiera de las propiedades 100,101,102,103,104 ó 105 cumple el predicado la función lógica se hace verdadera.

Ejemplo:

Si nosotros deseamos saber qué cuadros tienen puntos mayores de 2,200 al nivel del mar, usamos el siguiente predicado:

ALTURA = UNADE (190,195,MAYORQ,2200)

De la tabla #1 (ver Apéndice), vemos en curvas de nivel de la carta topográfica.

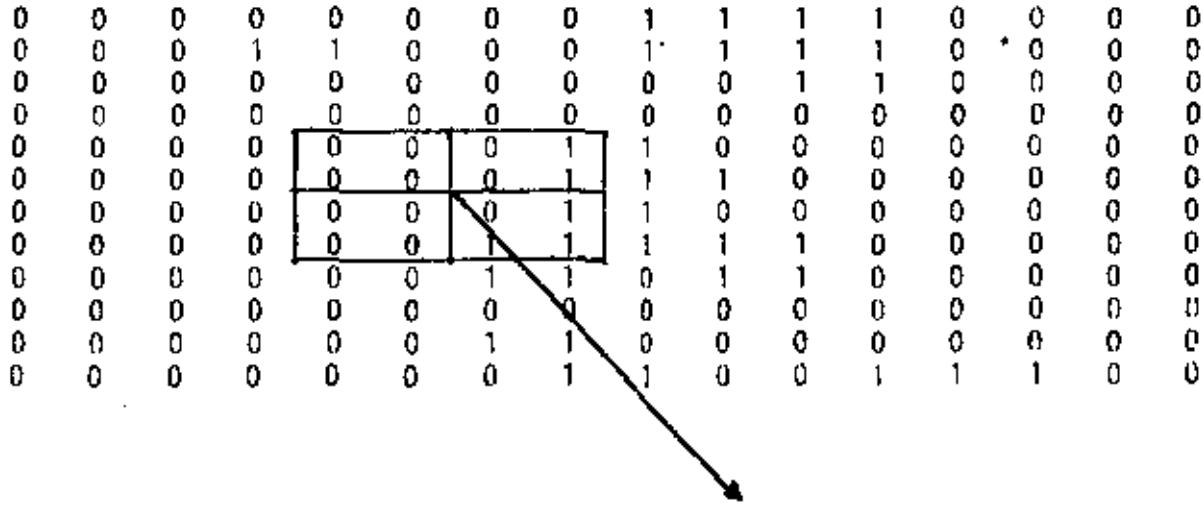
190 Nivel en la esquina izquierda superior
191 Nivel en la esquina derecha superior
192 Nivel en la esquina izquierda inferior
193 Nivel en la esquina derecha inferior
194 Nivel máximo
195 Nivel mínimo

Si el predicado se cumple para una sola propiedad ó sea que una sóla propiedad sea mayor de 2,200 la función lógica será verdadera.

Como se ve en la Figura 5, que pertenece a la carta topográfica, verificamos los resultados obtenidos y efectivamente vemos que en los subcuadros B y D del cuadro #20, hay curvas de nivel que exceden los 2,200 metros al nivel del mar, para el cuadro #19 no hay un sólo subucadro que tenga un nivel mayor de 2,200 mts. para el cuadro #27 sucede lo mismo y para el cuadro #28 hay 3 subcuadros, los B, C y D que exceden los 2,200 mts. al nivel del mar. (Esto lo vemos en la Figura 5 que pertenece a la carta topográfica).

```
LOGICAL FUNCTION ALTURA(N)
LOGICAL MAYORQ,UNADE
EXTERNAL MAYORQ
ALTURA=UNADE(190,195,MAYORQ,2200)
RETURN
END
```

```
LOGICAL ALTURA
EXTERNAL ALTURA
CALL BUSCA(ALTURA,4)
STOP
END
```



Equivalente a la figura 5

Listado 2

CUADRO #19 CUADRO #20

CUADRO #27 CUADRO #28

Fig. 5 Parte de la Carta Topográfica de Ojo Caliente, Zacatecas.

Aquí se han detectado regiones de más de 2 200 m de altura

PROP (A1, A2, A3, A4)

Es una función lógica que nos sirve para construir predicados. PROP es una función lógica que utiliza la función relacional Entre. Donde:

A1 Número de propiedad, en punto flotante
A2 Función relacional: por el momento, sólo es ENTRE
A3 Límite inferior, en punto fijo
A4 Límite superior, en punto fijo.

Ejemplo:

Si queremos un cuadro ó subcuadro que tenga un número de habitantes entre 150 y 500, definimos nuestro predicado en la siguiente forma:

PUEBLO = PROP (310., ENTRE, 150, 500)

Donde de la tabla #1 (ver Apéndice) sabemos que la prop. 310. es el número de habitantes.

Ahora, si queremos saber cuales son los subcuadros donde hay pueblos que tengan de 150 a 500 habitantes y no tengan carretera de más de dos carriles ó pavimentada ó terracería todo tiempo, ó terracería en tiempo de secas ó carretera federal, de cuota ó estatal, usamos PROP y UNADE en la forma siguiente:

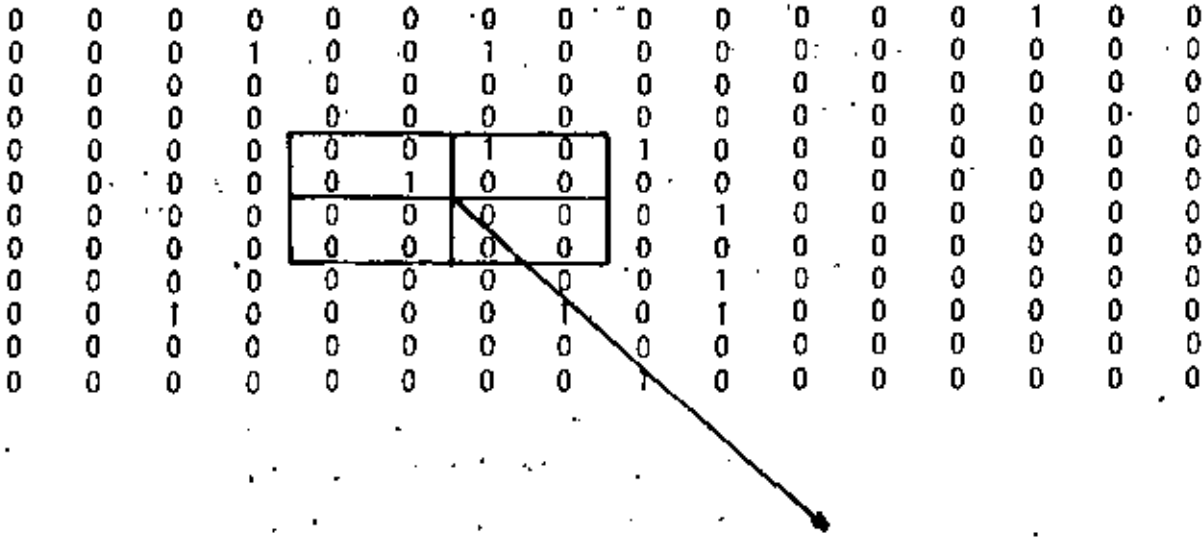
PUEBLO = PROP (310., ENTRE, 150, 500).AND. UNADE (110,116, MAYORQ,0)

Como se ve en la tabla #1 (ver Apéndice) las prop. 110 a 116 corresponden a los tipos de caminos indicados.

En la figura 6 , vemos en el subcuadro A del cuadro #19 que no existen pueblos que tengan habitantes entre 150 a 500, habitantes por lo tanto aparece un cero. Los subcuadros en los cuales nuestro predicado se hace verdadero son el subcuadro D del cuadro 19, en el cual existe un pueblo que tiene 500 habitantes y no tiene ningún tipo de camino, este pueblo se llama Las Coloradas. También en

```
LOGICAL FUNCTION PUEBLO (N)
LOGICAL ENTRE,PROP,UNADE,MAYORQ
EXTERNAL ENTRE,MAYORQ
PUEBLO=PROP(310.,ENTRE,150,500).AND.(.NOT.UNADE(110,116,MAYORQ,0))
RETURN
END
```

```
LOGICAL PUEBLO
EXTERNAL PUEBLO
CALL BUSCA (PUEBLO,4)
STOP
END
```



Equivalente a la figura 6

Listado 3

Fig. 6 Parte de la Carta de Uso del Suelo de Ojo Caliente, Zac.
Detección automática de pueblos que tienen entre 150 y 500 habitantes y están incomunicados. Se halló LAS COLORADAS en la parte superior, y LAS LAJAS en el cuadro 20.

el subcuadro A del cuadro #20 es verdadero el predicado, ya que existe un pueblo llamado Las Lajas, con 250 habitantes y no tiene ningún tipo de camino, la figura 6 pertenece a la carta de uso de suelo.

HAYVIA

Es una función lógica que consta de 2 argumentos como sigue:

HAYVIA (A1, A2)

Donde:

A1 Se ve en la tabla #5 y nos indica la clase de via de comunicación.

A2 Es el número del pueblo con el que se comunica y se ve en la tabla #4. (pág.111).

Por ejemplo, queremos saber en qué lugar hay un pueblo que se comunique con el pueblo llamado PANFILO NATERA, a través de carretera pavimentada. Si a nuestra función la llamamos VIAS, tenemos:

VIAS = HAYVIA (11,2019)

El 11 corresponde a una carretera pavimentada de la tabla #5.
El 2019 corresponde al pueblo llamado GRAL. PANFILO NATERA de la tabla #4.

Como vemos, en la Fig 7 en el cuadro 11 existe un pueblo que tiene carretera pavimentada a el pueblo GRAL. PANFILO NATERA y este pueblo se llama la ESQUINA DEL POTRERO, por lo tanto nuestra función se hace verdadera y aparece un uno en el listado 4.

SERPRO

Es una función lógica que nos sirve para la utilización de los servicios propuestos para la población, ésto se ve en la tabla #2, la estructura de SERPRO consta de 2 argumentos y es como sigue:

```
                        VIAS                DATE = 74122          08/51/19

      LOGICAL FUNCTION VIAS(N)
      LOGICAL HAYVIA
      VIAS=HAYVIA(11,2019)
C QUIERO SABER QUE PUEBLO TIENE CARRETERA PAVIMENTADA A  GRAL. PANFILO NATERA
      RETURN
      END
```

```
                        MAIN                DATE = 74122          08/51/19

      LOGICAL VIAS
      EXTERNAL VIAS
      CALL BUSCA(VIAS,3)
      STOP
      END
```

```
 0   0   0   0   0   0   0   0
 0   0   1   0   0   0   0   0
 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0
```

Listado 4

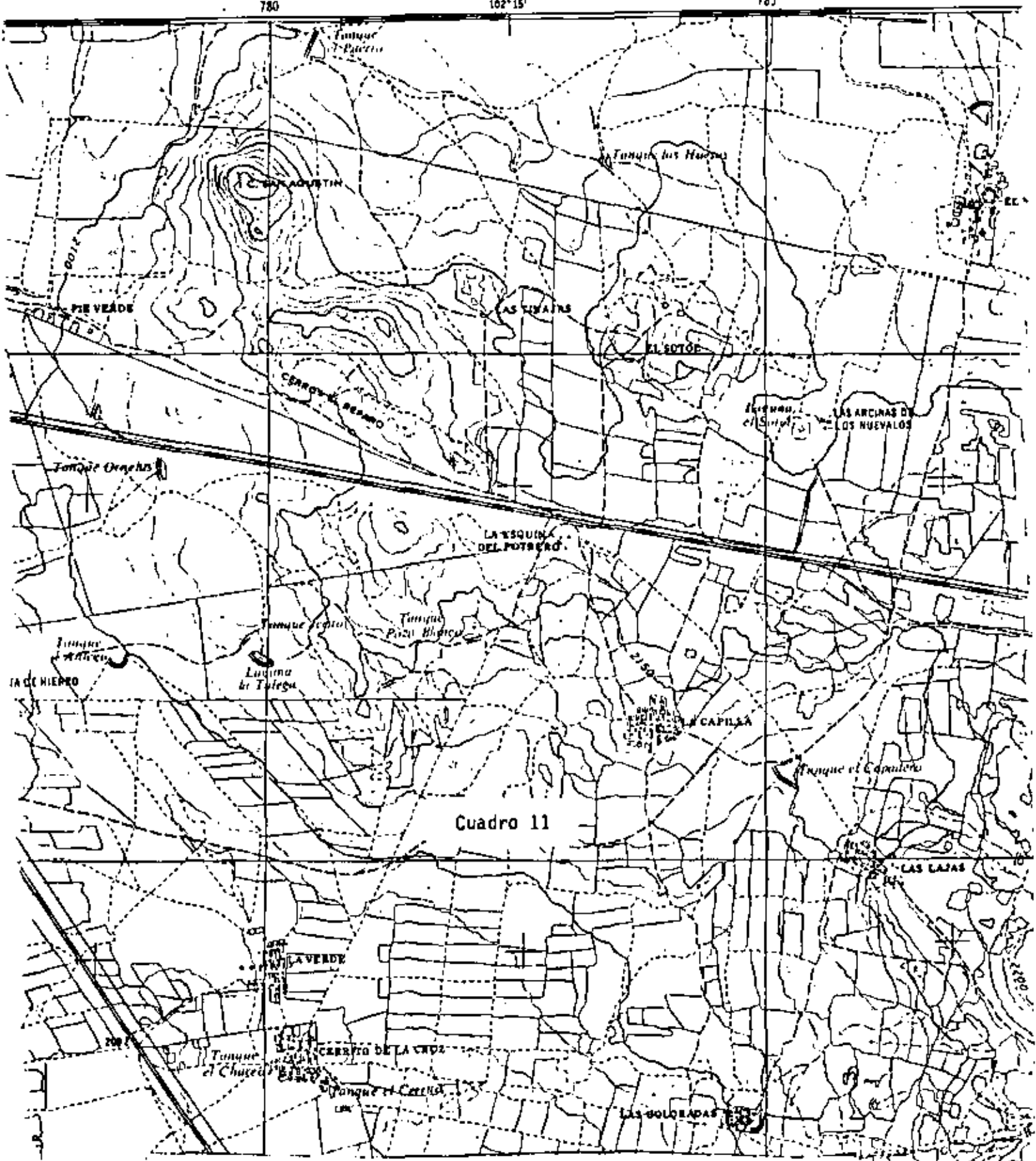
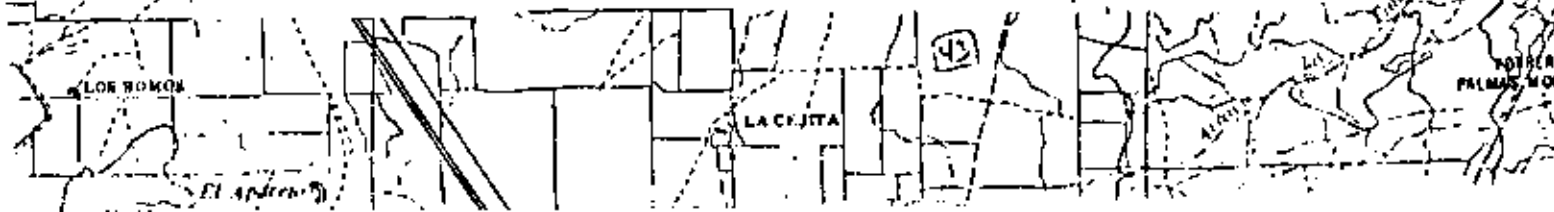


Fig. 7 Parte de la carta Topográfica de Ojo Caliente, Zac.



SERPRO (A1,A2)

Donde:

A1 Es el número para servicios para la población y va del 1 al 8 como sigue:

1. abastecimiento de agua de fuente superficial
2. abastecimiento de agua de fuente subterránea
3. escuela
4. telégrafo
5. drenaje por fosa séptica
6. drenaje por emisor
7. centro asistencial
8. energía eléctrica

A2 Es un número de entrada que, si es 1, nos indica que buscamos servicios A1 propuestos. Si es 0 nos indica que buscamos servicios del tipo A1 que no hayan sido propuestos.

En el ejemplo (ver listado 5) queremos saber en qué lugares se han propuesto escuelas, si llamamos ESC a nuestra función queda de la siguiente forma:

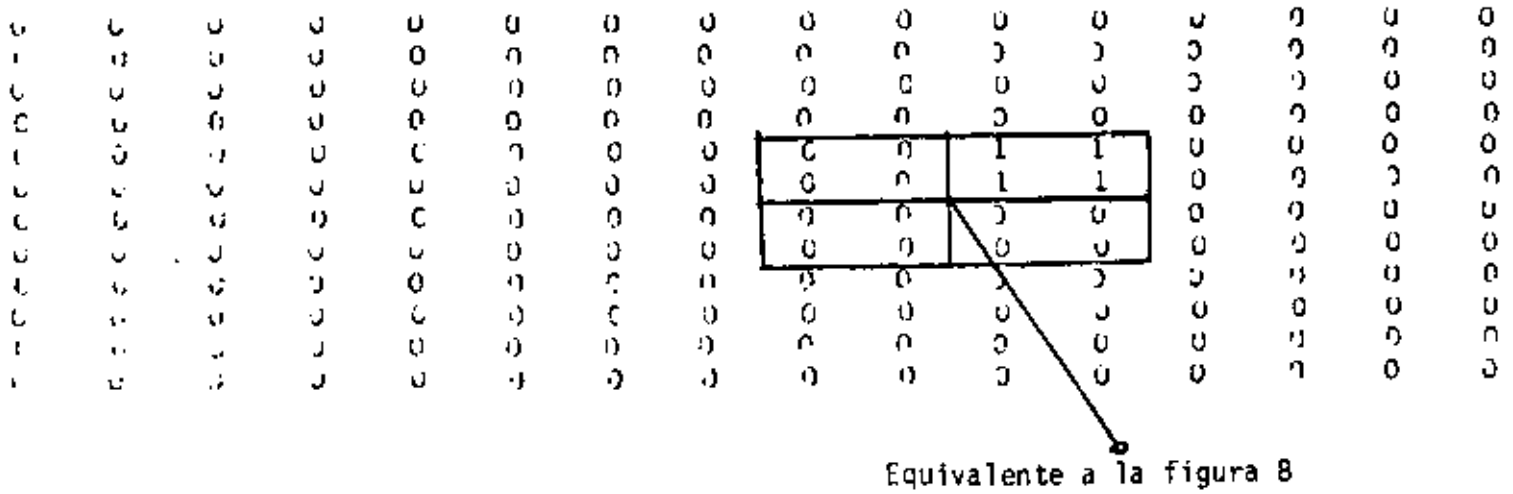
ESC = SERPRO (3,1)

En HAYVIA, SERPRO, SERPOB y PUEBLO, sólamente tienen significado para cuadros porque así se definió en el Banco de Datos Geográficos.

Como se comprueba en la Figura 8 vemos que en el cuadro #22 hay un 3 que nos indica que se ha propuesto una escuela y se ve en el listado 5 de computadora que aparecen 1's en los subcuadros del cuadro #22 y ésto es debido a que SERPRO por definición del Banco de Datos Geográficos entró a nivel de cuadro.

```
FRAN IV G LEVEL  21          ESC          DATE = 74123          08/22/00
      LOGICAL FUNCTION ESC(N)
      LOGICAL SERPRO
      ESC=SERPRO(3,L)
C EN QUE LUGARES HAN PROPUESTO ESCUELAS
C L ES ESPROPUESTO, 0 ES NO PROPUESTO
      RETURN
      END
```

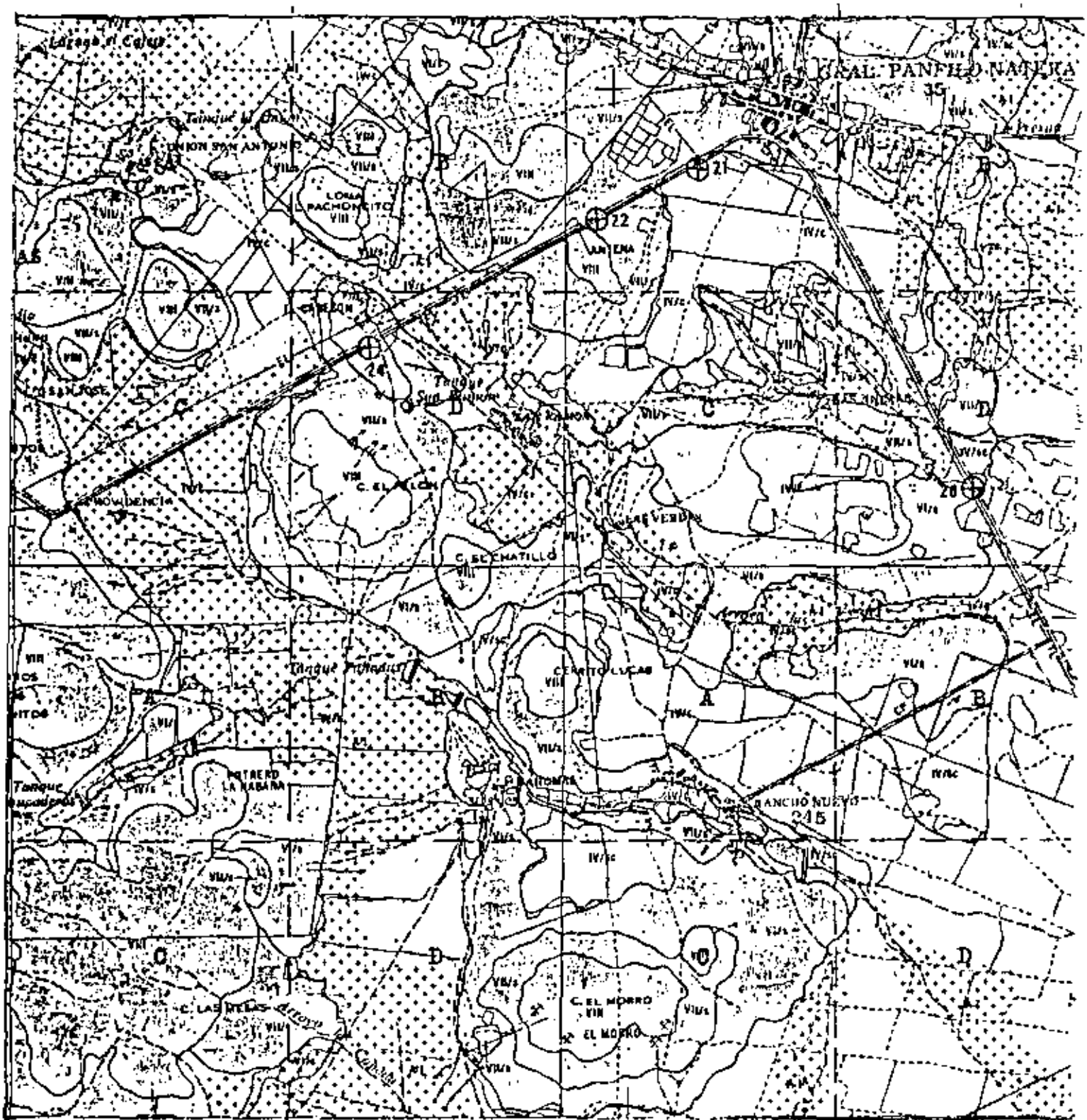
```
FRAN IV G LEVEL  21          MAIN         DATE = 74123          08/22/0
      LOGICAL ESC
      EXTERNAL ESC
      CALL BUSCA(ESC,4)
      STOP
      END
```



Equivalente a la figura 8

CUADRO #21

CUADRO #22



CUADRO #29

CUADRO #30

Fig. 8 Parte de la Carta de Uso Potencial de Ojo Caliente, Zacatecas.

¿EN QUE PUEBLOS SE HAN PROPUESTO ESCUELAS?

PUEBLO

Es una función lógica que consta de 1 argumento como sigue:

PUEBLO (Arg)

Donde:

ARG.- es el equivalente numérico de un pueblo y esto se ve en la tabla #4.

Esta función nos sirve para localizar en qué lugar se encuentra un pueblo buscado. Ejemplo:

En la zona de Ojo Caliente, Zac. deseamos buscar en qué cuadro está el pueblo de San Pablo.

1o. Construímos nuestra función en este caso le llamamos PABLO.

```
LOGICAL FUNCTION PABLO (N)
LOGICAL PUEBLO
PABLO = PUEBLO (2037)
RETURN
END
```

Como se puede observar el número 2037 en la tabla #4 corresponde al pueblo de San Pablo.

2o. Construímos el programa principal y buscamos a nivel de cuadro, Ejemplo:

```
LOGICAL PABLO
EXTERNAL PABLO
CALL BUSCA (PABLO, 3)
STOP
END
```

El resultado es el siguiente:

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
```

Como podemos comprobar en la Figura 8' que corresponde a una parte de la Carta Edafológica de Ojo Caliente, Zacatecas, existe un pueblo llamado San Pablo en el cuadro 38 y en este cuadro es donde nuestra función se hace verdadera y existe un uno.

SERPOB

Es una función lógica que consta de 8 argumentos (ver tabla #3) y son los siguientes:

SERPOB (NUM, a,b,c,d,e,f,g)

NUM.- Es el número que está en la tabla #3 que corresponde a los servicios de la Población y va del 1 al 9 en la siguiente forma (se localizan en la Carta de Uso del Suelo):

1. Abastecimiento de agua
2. Medio de almacenamiento
3. Forma de distribución
4. Drenaje
5. Asistencial
6. Municipal
7. Educacional
8. Corriente eléctrica
9. Comunicaciones.

a. Es una subdivisión de los servicios en la población, por ejemplo: si tenemos,

   1a. Indica que tenemos abastecimiento de agua por manantial (ver tabla #3).

b. Es una subdivisión de los servicios de la población; por ejemplo: si tenemos,

   8b Indica que tenemos corriente eléctrica por medio de planta propia (ver tabla #3 al final de este reporte).

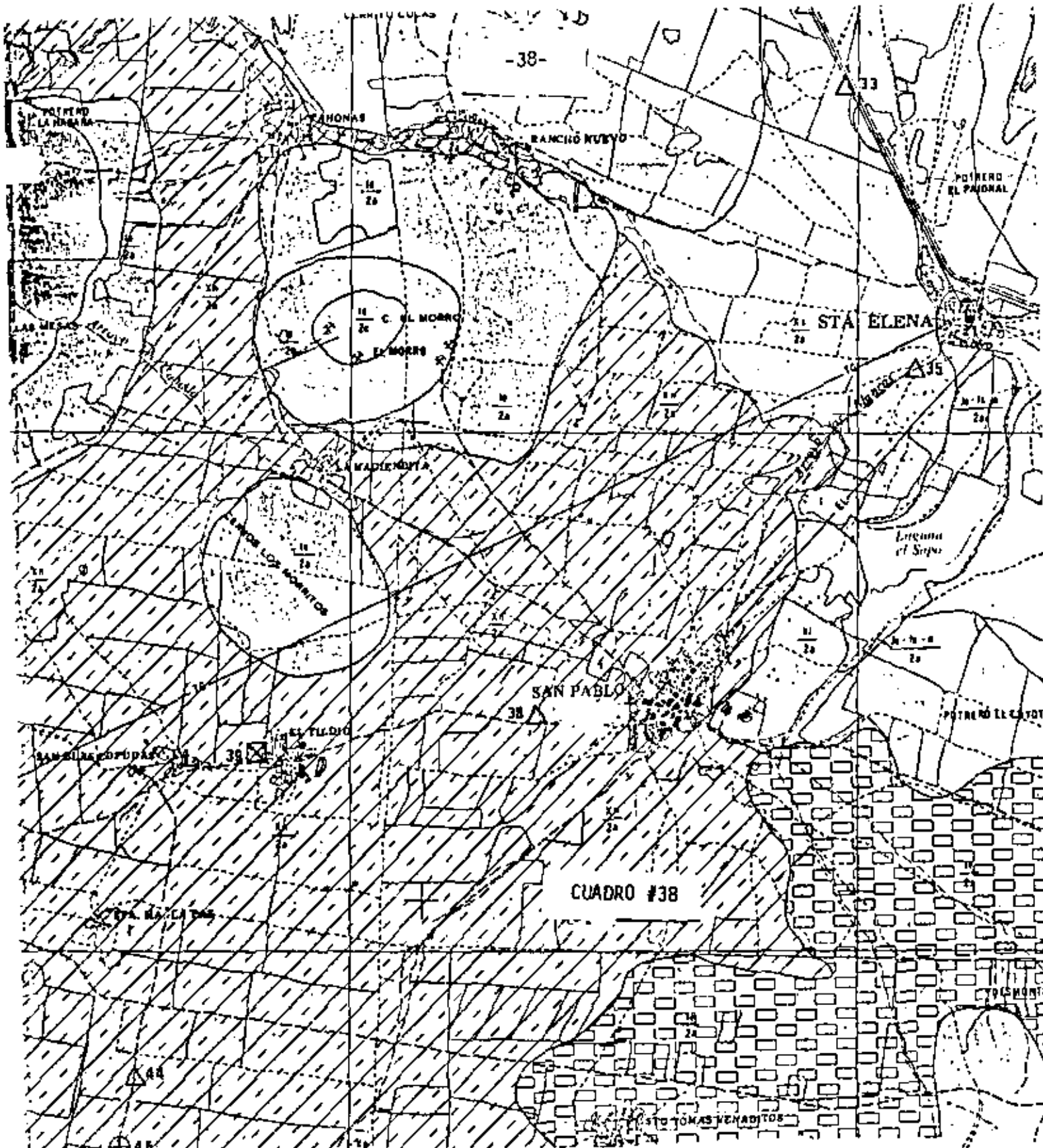


Fig.8 Parte de la Carta Edafológica de Ojo Caliente, Zac.

c. Es una subdivisión en los servicios de la población, por ejemplo: si tenemos,

7c Indica que existe Educacional a nivel de secundaria (ver tabla #3)

d. Es una subdivisión en los servicios de la población, por ejemplo: si tenemos,

3d indica que tenemos forma de distribución por medio de tracción animal. (ver tabla #3)

e. Es una subdivisión en los servicios de la población, por ejemplo: si tenemos,

9e Indica que tenemos comunicaciones por medio de radio difusora (ver tabla #3).

f. Es una subdivisión en los servicios de la población, por ejemplo: si tenemos,

1f Indica que tenemos abastecimiento de agua por medio de pozo (ver tabla #3).

g. Es la última subdivisión en los servicios de la población, por ejemplo: si tenemos,

7g Indica que tenemos educacional a nivel de enseñanza superior (ver tabla #3).

Ahora podemos tener el siguiente ejemplo:

1fg-2b-3ab-4abc-5b-6a-7abcdefg-8a-9bc

Este ejemplo nos indica que deseamos una población con los siguientes servicios:

1fg	Abastecimiento de agua de pozo y poza
2b	Medio de almacenamiento de tanque elevado
3ab	Forma de distribución por tubería y canal
4abc	Drenaje por emisor por fosa séptica y fosa
5b	Asistencial con clínica
6a	Municipal con rastro
7abcdefg	Educacional con pre-primaria, primaria, secundaria, preparatoria, normal, enseñanza técnica y enseñanza superior.
8a	Corriente eléctrica por línea
9bc	Comunicaciones por telégrafo y teléfono

Para el uso del Banco de Datos Geográficos las subdivisiones de los servicios de la población tienen un equivalente que es el siguiente:

a	-	1
b	-	4
c	-	16
d	-	64
e	-	256
f	-	1024
g	-	4096
NADA	-	0

Si por ejemplo, deseamos buscar lugares que tengan educacional a nivel de secundaria que es 7c, y sabemos que c es 16 en SERPOB, lo hacemos de la siguiente forma:

SERPOB (7,0,0,16,0,0,0,0)

Como se observa el primer argumento sigue conservando el número de la tabla #3, en el segundo argumento tenemos cero ya que no estamos preguntando por pre-primaria, también el segundo argumento es cero, ya que no preguntamos por escuela primaria, pero para el tercer argumento tenemos 16, ya que estamos preguntando por un lugar donde existe secundaria y para el resto de argumentos tenemos ceros.

Es muy importante que la función tenga siempre sus ocho argumentos, también podemos alterar el órden de los argumentos a excepción del primero. Ejemplo:

SERPOB (7,0,0,16,0,0,0,0) es equivalente a SERPOB (7,16,0,0,0,0,0,0)

En un ejemplo por computadora (ver listado 5'), deseamos saber en qué subcuadros de la zona de Ojo Caliente, Zac. hay abastecimiento de agua solo por medio de pozo.

1. Construímos nuestra función lógica y la llamamos POZO en la forma siguiente:

```
LOGICAL FUNCTION POZO (N)
LOGICAL SERPOB
POZO = SERPOB (1, 1024, 0,0,0,0,0,0)
RETURN
END
```

Como observamos en la tabla #3 el 1 corresponde a Abastecimiento de Agua y el 1024 corresponde a f y f en la tabla #3 corresponde a pozo, como se observa el órden de los últimos 7 argumentos no importa.

2. Construímos el programa principal y buscamos a nivel de subcuadro.

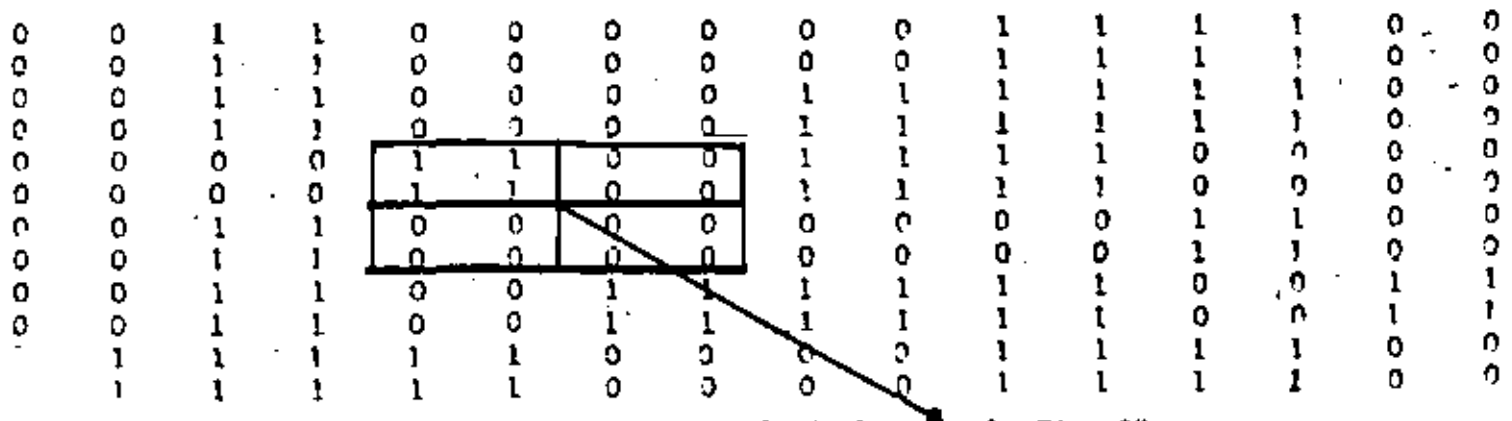
```
LOGICAL POZO
EXTERNAL POZO
CALL BUSCA (POZO, 4)
STOP
END
```

En el listado 5' podemos ver el resultado a nivel de subcuadro.

En la figura 8" que es una parte de la Carta de Uso del Suelo de Ojo Caliente, Zac., vemos que en los cuatro subcuadros del cuadro 19 aparecen únicamente unos, que nos indican que en ese cuadro es nuestra función verdadera, en el subcuadro 0 del cuadro 19, vemos que el pueblo llamado Las Coloradas tiene 1f, que es abastecimiento de agua por medio de pozo, también en ese mismo subcuadro el pueblo La Cejita

```
FAN IV G LEVEL  21                POZO                DATE = 74126          07/47/51

      LOGICAL FUNCTION POZO(N)
      LOGICAL SERPOB
      POZO=SERPOB(1,1024,0,0,0,0,0,0)
C QUIERO SABER DONDE HAY UN LUGAR QUE EN SERVICIOS DE LA POBLACION
C TENGA ABASTECIMIENTO DE AGUA POR MEDIO DE UN POZO
      RETURN
      END
```

```
FAN IV G LEVEL  21                MAIN                DATE = 74126          07/47/5

      LOGICAL POZO
      EXTERNAL POZO
      CALL BUSCA(POZO,4)
      STOP
      END
```



Equivalente a la Fig. 8"

Listado 5'

CUADRO #19 CUADRO #20

CUADRO #27 CUADRO #28

Fig. 8ª Parte de la Carta de Uso Del Suelo de Ojo Caliente, Zac. Se han encontrado múltiples lugares con abastecimiento de agua por POZO.

tiene 1F-3e-7b de servicios de la población; en este caso el 1f está contenido como la función SFRPOB tiene contenido a nivel de Cuadro, todos los subcuadros de este cuadro se hacen verdaderos;en los cuadros 20, 27 y 28 no existen pueblos que como servicios de la población tengan abastecimiento de agua por medio de pozo, por esta razón, isten solamente ceros en estos subcuadros.

EVALUA

45

Nos sirve para hacer una evaluación de propiedades de una región.

VALOR

Nos sirve para determinar cantidades numéricas referentes a las props. de la tabla #1 (ver Apéndice) de cualquier región, ya sea en M^2 ó su valor en cualquier unidad, etc.
Ejemplo:

VALOR (310.) nos da el valor de la propiedad 310, es decir, el-número de habitantes.

El valor de una propiedad es 0 si ésta no existe.

FUNCIONES NUMERICAS

Hasta ahora para buscar zonas de interés hemos usado predicados, es decir, que existen (1) o no existen (0) en determinado lugar. Empero, es a veces conveniente usar funciones cuyo rango no es sólo el conjunto (0, 1), sino un conjunto más poblado, digamos (0, 100). Por ejemplo, si quiero hallar un lugar RICO puedo calificar cada lugar con números de 0 a 100 donde 0 nos indique "paupérrimo" y 100 nos indique "muy rico", pero donde también sean permisibles todos los resultados intermedios, con la interpretación correspondiente en la escala de riqueza: es decir, 10 es "bastante pobre", 52 "regular", etc.

A estas propiedades que tienen como rango un conjunto de varios valores que representan "grados" o "intensidades" de una propiedad se nos ha dado en llamarles propiedades numéricas, para diferenciarlas de los predicados,donde sólo hay 2 valores de salida.

Continuaremos nuestro ejemplo diciendo que los lugares que tienen minas de oro ó plata son ricos, los que tienen pantanos ó zonas erosionadas son pobres, y los que tienen pastizales naturales ó cultivos son "regulares". Como ahora podemos dar más precisión a nuestras interpretaciones y como el oro vale 3 veces más que la plata, diremos que una mina de plata en un lugar contribuye con 100 puntos (el número 100 es arbitrario, pero sirve de referencia para los demás números ó coeficientes, que guardan una relación ó proporción con éste de la plata) a la riqueza de su lugar, mas una mina de oro contribuye con 300 puntos. Un pantano contribuye con -150 puntos (empobrece el lugar), mas una zona erosionada contribuye tan solo con -50 puntos (lo empobrece menos). Es decir,

RIQUEZA = 300 *(número de minas de oro) + 100 (número de minas de plata) -150 *(porcentaje de terreno pantanoso) -50 * (porcentaje de terreno erosionado)

Debe aclararse que los coeficientes 300, 100, -150, etc.,son "subjetivos" en el sentido de que no hay reglas exactas para hallarlos, por lo que por lo general varían; personas distintas asignarán pesos ó coeficientes distintos. También el incluír una propiedad (cañaverales, por ejemplo) depende del usuario y del fin con que se vaya a utilizar la función numérica RIQUEZA que se está definiendo.

Al evaluarse RIQUEZA en toda una zona, nos dará un arreglo de números. Estos mientras más grandes sean más RICO será el lugar que los posee.

LAS FUNCIONES EVALUA, EVALU, EVAL

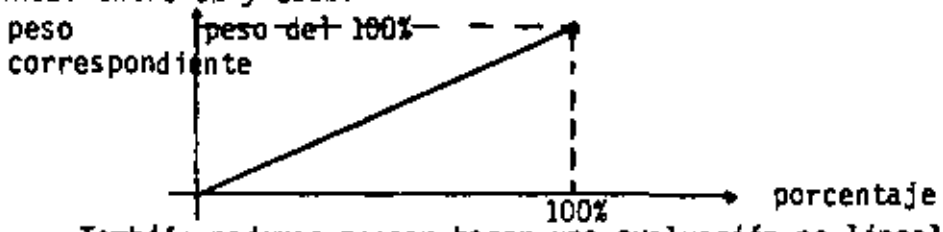
Para lo anterior se usa la función EVALUA, con sus variantes EVALU y EVAL, a la que se le proporcionan las propiedades que deberán tomarse en cuenta, junto con los coeficientes o pesos que se les asigne. En el ejemplo, se le proporciona la siguiente información:

INFORMACION PARA EVALUA

PROPIEDAD #	PESO PARA 100%
minas de oro 172*	300
minas de plata 173**	100
pantanos 185	-150
terreno erosionado 286	-50

* y ** suponemos que esos números son sus equivalentes

Entonces, si el 100% del área de un lugar es terreno erosionado, se contribuye con -50 puntos; pero si solo el 30% está erosionado, la contribución es -50 x .3 ó sea -15 y si no hay erosión alguna, la contribución es 0. Es decir, se hace una interpolación lineal entre 0% y 100%.



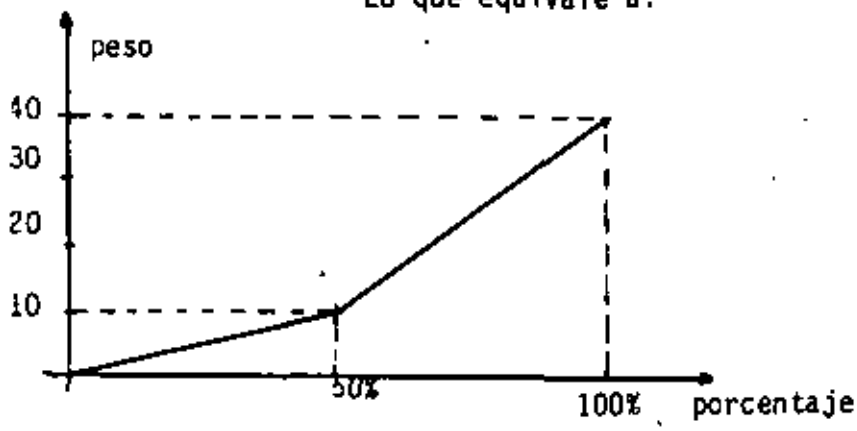
También podemos pensar hacer una evaluación no lineal, especificando un peso para el 50% no necesariamente igual a la mitad del peso correspondiente para 100%.

Por ejemplo si para cierto fín el porcentaje de mezquital importa poco mientras no llegue al 50%, pero después importa bastante más, podemos fijarle 40 al porcentaje 100%, pero sólo 10 en lugar de 20 al porcentaje = 50% como sigue:

INFORMACION ALTERNA PARA EVALUA

PROPIEDAD #	PESO PARA 0%	PESO PARA 50%	PESO PARA 100%
287	0	10	40

Lo que equivale a:



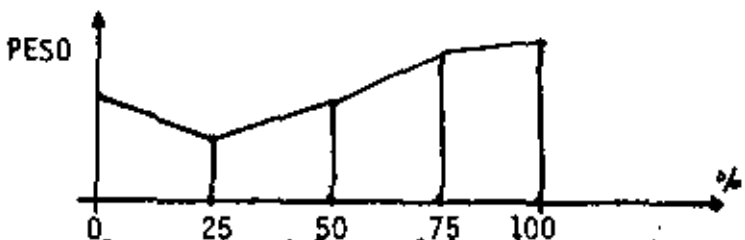
Nótese que se sigue interpolando linealmente entre 0 y 50 y también entre 50 y 100.

Finalmente, existe una tercera forma de dar la información incluyendo asímismo los valores (pesos) en 0%, 25%, 50%, 75% y 100%.

INFORMACION ALTERNA PARA EVALUA

PROPIEDAD #	PESO PARA 100%	PESO PARA 75%	PESO PARA 50%	PESO PARA .25%	PESO PARA 0%

lo que permite hacer la curva más no-lineal:



Ejemplo:

Vamos a evaluar potencial económico de zonas boscosas, suponiendo que los cedros (propiedad 243.) rojos valen lo doble que los eucaliptos (propiedad 240) y éstos valen lo triple que los pirules (propiedad 245), a lo que les daremos el valor de 1. Digamos también que los chaparrales (propiedad 268) son "nocivos" por lo que valen -1. Todo lo demás no importa (vale 0).

La información es:

	Propiedad	Valor al 100%
PIRULES	245	1
EUCALIPTOS	240	3
CEDROS	243	6
CHAPARRALES	268	-1

Esta información se mete a un arreglo entero de dimensiones 4x2 mediante el postulado:

INTEGER BOSQUE (4,2)/245,240,243,268,1,3,6,-1/

El nombre del arreglo puede ser cualquiera, de 1 a 6 letras.

Este arreglo se le suministra a nuestro sistema llamando a la función DEFINE, a la que también se le comunican las dimensiones del arreglo. Esto es con el fin de que el sistema pueda deducir si se le están proporcionando pesos del 100%, del 100 - 75 - 50 - 25 - 0%, ó del 100 - 50 - 0%.

El postulado es CALL DEFINE (BOSQUE, 4, 2) y finalmente llamamos a EVALUA con los siguientes argumentos:
El 1er. argumento es el nivel 3 en este caso (cuadros).
El 2o. es el predicado de necesidad, es decir, que sólo se evaluarán los lugares donde este predicado se hace cierto. Como en nuestro caso, queremos evaluar todos los cuadros, hacemos este predicado = .TRUE.
El 3o. es la función entera a sumar, es decir, que esta función se evaluará también en el lugar, y este valor se sumará a la evaluación hecha con las propiedades y pesos dados en BOSQUE para darnos la evaluación total de cada lugar. Como en nuestro caso no deseamos agregar algo más a lo que BOSQUE nos dé, definimos esta función entera ISUMA=0.
El 4o. es siempre el número 1, que nos indica que siempre imprimiremos los resultados de la evaluación en cada punto evaluado. Otras alternativas, no implementadas son:

2 → imprimir los máximos
4 → imprimir los mínimos, etc.

El programa total es:

```
      Logical function prnece (n)
      prnece = .true.                    DEFINICION DEL
                                         PREDICADO
      return                             PRNECE
      end

      Integer function ISUMA (n)
      ISUMA = 0                          DEFINICION DE LA
                                         FUNCION ENTERA ISUMA
      return
      end
```

```
External prnece, isuma
logical prnece
integer isuma                                          PROGRAMA
integer BOSQUE (4,2)/245,240,243,268,1,3,6,-1/         PRINCIPAL
call define (BOSQUE, 4,2)
call evalua (3,PRNECE,ISUMA,1)
stop
end
```

Nótese el uso de external, logical e integer.

Variaciones del arreglo que contiene los pesos

Si yo quisiese usar un conjunto de coeficientes más completo y con no-linealidades para la definición de la función de la zona boscosa, por ejemplo:

PROPIEDAD #	PESO PARA 100%	PESO PARA 75%	PESO PARA 50%	PESO PARA 25%	PESO PARA 0%
245	1	1	1	1	0
240	3	2	2	1	1
263	6	5	4	3	0
268	-1	-1	-1	0	0

todo lo anterior sigue igual, cambiando únicamente el INTEGER.
y el CALL DEFINE como sigue:

```
INTEGER BOSQUE (4, 6)/245,240,263,268,1,3,6,-1,1,2,5,-1,   1,2,4,-1,1,
X1,3,0,0,1,0,0 /
CALL DEFINE (BOSQUE, 4, 6)
```

(Aquí vemos además el uso de una X en la columna 6 de la tarjeta cuando la línea que lo contiene es continuación de la anterior)

Reglas

La primera dimensión del arreglo debe ser igual al número de propiedades a ser evaluadas. 4 en nuestro caso. El máximo es 20.

Si la 2a. dimensión es 2, el arreglo contendrá las 2 columnas siguientes:

PROPIEDAD # PESO PARA 100%

Se supone que el peso para 0% es 0, y se hace una interpolación lineal para pesos intermedios.

Si la 3a. dimensión es 3, el arreglo contiene:

PROPIEDAD # PESOS PARA 100% PESO PARA 0%

Si la 2a. dimensión es 4, el arreglo contiene:

PROPIEDAD# PESOS PARA 100% PESO PARA 50% PESO PARA 0%

Si la 2a. dimensión es 5, es un error.

Si la 2a. dimensión es 6, el arreglo contiene:

	P E S O S P A R A				
PROPIEDAD #	100%	75%	50%	25%	0%

Es decir, sólo son válidos para la segunda dimensión los números 2, 3, 4, y 6, con las interpretaciones correspondientes.

También es válido un 1 como 2a. dimensión, significando que el arreglo no se usa. De esto veremos un ejemplo después (obtención de área total en metros cuadrados). Si no se usa el arreglo, no es necesario definirlo.

USO DEL PREDICADO DE NECESIDAD.

Si por alguna razón no deseo evaluar todos los lugares, por ejemplo, porque tome mucho tiempo de máquina ó porque sé de antemano que sólo ciertos lugares me interesa evaluar, uso el predicado de necesidad para indicar que sólo en aquellos lugares en que el predicado se satisfaga, se hará la evaluación; en los lugares donde no hubo evaluación por ser el predicado falso en ellos aparecerá un 0 como "evaluación".

Por ejemplo, si sólo me interesa evaluar zonas boscosas en regiones superiores a 1,000 metros, uso los 3 programas anteriores (donde se definieron a PRNECE, ISUMA y al arreglo BOSQUE) con el único cambio siguiente: la nueva definición de PRNECE es:

```
LOGICAL FUNCTION PRNECE (N)
LOGICAL PRO, MAYORQ
EXTERNAL MAYORQ                                  DEFINICION DEL
PRNECE = PRO (195., MAYORQ, 1000)                NUEVO PREDICADO
return                                           DE NECESIDAD
end
```

Con ese cambio indicamos que es necesario poseer una altura mínima superior a 1,000 metros para proceder a la evaluación de un lugar.

USO DE LA FUNCION ENTERA A SUMAR

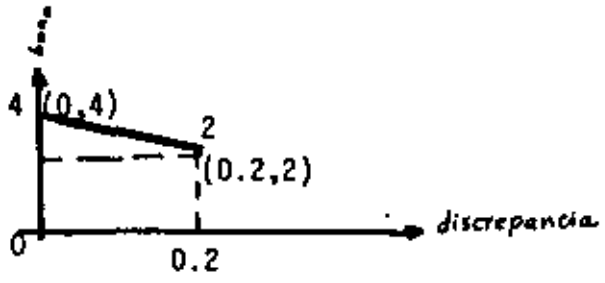
Para evaluación de funciones más complejas, que no se puedan expresar en polinomios lineales con el arreglo de pesos, se usa la función entera a sumar. El valor de esta función se agrega al valor obtenido mediante el arreglo de pesos, para así obtener la evaluación del lugar. Para esto a menudo se usa la función entera VALOR (prop.), cuyo resultado es el valor de la propiedad prop.

Por ejemplo, supongamos que en el ejemplo de evaluación de zonas boscosas, me "atrae" la propiedad "que haya igual número de coníferas (224.) que de pastizales (210.)"

Más específicamente, si la discrepancia entre la extensión de coníferas y la de pastizales es menor al 20% del área conjunta de ambos, estoy dispuesto a darle al área un "bono" adicional en su evaluación, que va de 4 a 2; 4 cuando la discrepancia no exista y 2 cuando sea de un 20%. Si la discrepancia es mayor de un 20%, no hay bono adicional.

Esta propiedad no se podría computar con el arreglo de pesos; pero sí se puede hacer mediante la siguiente definición de la nueva función entera a sumar:

```
      INTEGER VALOR
      INTEGER FUNCTION ISUMA (N)
      ISUMA = 0
      IDIFE = VALOR (224.) - VALOR (210.)
      ITOTAL = VALOR (224.) + VALOR (210.) + 1
      DISCR = IDIFE / ITOTAL
      IF (DISCR .GT. 0.2) RETURN
c     Si la discrepancia es mayor de 0.2 regresa 0 como bono
c     pero si no lo es, regresa un bono entre 2 y 4 menor mientras
c     mayor sea la discrepancia
      ISUMA = 4. - 10. * DISCR
      RETURN
      END
```



Ahora sí, si usamos EVALUA con esta nueva función entera a sumar ISUMA y con el nuevo predicado PRNECE, estaremos evaluando regiones boscosas arriba de mil metros tomando en cuenta pirules, eucaliptos, cedros y chaparrales, más la discrepancia entre coníferas y pastizales.

USOS DE EVALU Y EVAL.

Cuando la función entera a sumar es cero, es más fácil usar EVALU en vez de evalúa:

```
EVALU (nivel, PRNECE, 1) = EVALUA (nivel, PRNECE, función a sumar hecha 0, 1)
```

Cuando el predicado de necesidad es TRUE., ó sea que vamos a evaluar en todos los lugares, es más fácil usar EVAL que evalúa:

EVAL (nivel, función a sumar, 1) = EVALUA (nivel, predicado de necesidad hecho .TRUE., función a sumar, 1)

Por ejemplo, si deseamos conocer el área total de cultivo de cada cuadro en miles de metros cuadrados, creamos nuestro predicado como sigue:

ISUMA = 62500 * VALOR (100.)

Aquí obtenemos el área total en metros cuadrados, el factor es 62500, el cual es arbitrario y se puede dar en las unidades que se quieran; como un subcuadro tiene un área de 6.25 Km^2, para obtener metros cuadrados multiplicamos por 6 250000. Nótese aquí que sólo se usó ISUMA y no se necesitó el arreglo con pesos. El factor es 62 ya que el resultado en miles de $mt.^2$ y el valor de la propiedad (674) está entre 0 y 100.

Como se verifica en la Fig. número 9 de la carta Edafológica, vemos que la cantidad de Cambisol Eútrico como dominante representado en el carta Edafológica como Be está en la siguiente distribución:

SUBCUADRO	CUADRO	AREA TOTAL
a	19	5890000 m^2
b	19	279000 "
c	19	5580000 "
d	19	1860000 "
a	20	0 "
b	20	1860000 "
c	20	0 "
d	20	0 "
a	27	1240000 "
b	27	620000 "
c	27	0 "
d	27	0 "
a	28	0 "
b	28	0 "
c	28	0 "
d	28	0 "

```
FTN IV G LEVEL 21          ISUMA          DATE = 74094          08/
      INTEGER FUNCTION ISUMA(NI
      INTEGER VALOR
C QUIERO SABER EL AREA TOTAL EN CADA SUBCUADRO DE CAMBISOL EUTRICO
C EN METROS CUADRADOS COMO DOMINANTE X1000
      ISUMA=62*VALOR (674.)
      RETURN
      END
```

```
FTN IV G LEVEL 21          MAIN           DATE = 74094          08/1
      EXTERNAL ISUMA
      DIMENSION IAR(1,1)
      CALL EVAL(4,ISUMA,1)
      STOP
      END
```

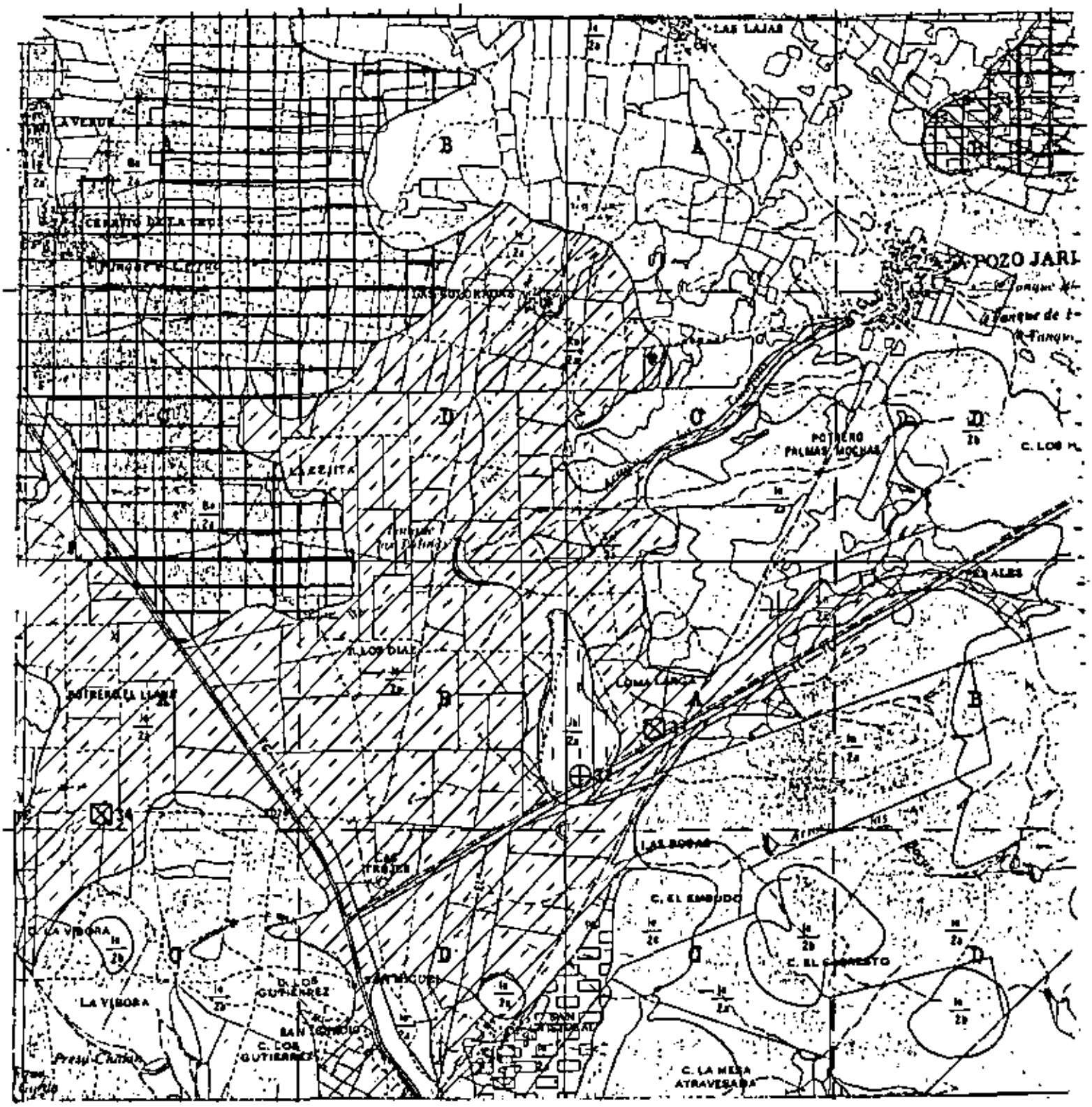
Listado 6 (primera parte)

6200	6200	6200	4340	2480	2170	2790	62	496	0	0	0	0	0	0
6200	6200	6138	3720	1240	1860	4340	4340	3596	0	0	0	0	0	0
6200	6138	6138	4030	2170	124	1860	3720	4030	310	0	0	0	0	0
6200	6138	6200	4960	3100	0	496	4030	3844	1240	0	0	0	0	0
0	2790	6200	6076	5890	2790	0	1860	3410	310	0	0	0	0	0
0	0	1860	2450	5580	1860	0	0	0	0	0	0	0	0	0
0	0	0	0	1240	620	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Equivalente a la figura 9

Listado 6
(segunda parte)

CUADRO #19 CUADRO #20



CUADRO #27 CUADRO #28

Fig. 9 Parte de la Carta Edafologica de Ojo Caliente, Zacatecas
En cada subcuadro, se ha hallado el area total de CAMBISOL EUTRICO.

En la Fig. número 9 de la carta Edafológica, el cambisol Eútrico como dominante se presenta como Be por lo cual, para mayor identificación se le puso cuadrícula. Esto se hizo a nivel de subcuadro.

LA FUNCION CERCA

CERCA es una función que tiene 4 argumentos y nos sirve para relacionar propiedades de diferentes ramas:

CERCA (PRO1,PRO2,DIS,NIVEL)

Donde:

PRO1	es el primer arreglo de la función de relación
PRO2	es el segundo arreglo de la segunda función que se va a comparar
DIS	es la distancia deseada
NIVEL	es el nivel al que buscamos

CERCA es una función que nos elimina la independencia entre cuadros y subcuadros.

Es muy importante ya que si nosotros queremos localizar una región que está entre cuadros y subcuadros se hace por medio de CERCA.
Ejemplo usando CERCA:

Deseamos saber donde hay minas que no disten más de 10Kms. de cualquier tipo de camino.

Primero, definimos nuestra función que busca minas como sigue:

```
      LOGICAL FUNCTION MINA (N)
      LOGICAL MAYORQ, PRO
      EXTERNAL MAYORQ
      MINA = PRO (170., MAYORQ, 0)
      RETURN
      END
```

Segundo, definimos nuestra función que busca caminos y la llamamos camino y queda como sigue:

```
LOGICAL FUNCTION CAMINO (N)
LOGICAL MAYORQ, UNADE
EXTERNAL MAYORQ
CAMINO = UNADE (110, 116, MAYORQ, 0)
RETURN
END
```

Tercero, definimos el predicado que relaciona los resultados de ambas funciones a través de la función CERCA en la forma siguiente:

```
LOGICAL FUNCTION MIAL (N)
LOGICAL CERCA
INTEGER*2 MINAS (12,16), ICARRE (12,16)
COMMON MINAS, ICARRE
MIAL = CERCA (MINAS, ICARRE, 1,3)
RETURN
END
```

En la función que llamamos MIAL, vemos que existe un INTEGER*2 MINAS (12,16) el cual define los tamaños de nuestros arreglos. Como el tamaño máximo es de (12,16) para subcuadros lo ponemos ya que así tenemos la opción de usar los niveles 3 y 4. En la siguiente línea de la función MIAL tenemos COMMON MINAS, ICARRE la cual nos sirve para comunicar los valores obtenidos en esta función MIAL con el programa principal; como buscamos a un cuadro vecino (1cuadro = 5Km. de longitud) ponemos un 1 y un 3 por el nivel en MIAL = CERCA (MINAS, ICARRE,1,3). El programa principal queda:

```
INTEGER*2 MINAS (12,16), ICARRE (12,16)
LOGICAL MINA, CAMINO, MIAL
EXTERNAL MINA, CAMINO, MIAL
COMMON MINAS, ICARRE
CALL IBUSCA (MINA, 3, MINAS)
CALL IBUSCA (CAMINO, 3, ICARRE)
CALL IBUSCA (MIAL, 3, MINAS)
STOP
END
```

En el programa principal observamos que tenemos una nueva rutina llamada IBUSCA que trabaja igual que la rutina anteriormente definida llamada BUSCA, con la diferencia de que IBUSCA tiene un argumento más en el cual deposita los resultados obtenidos, esta función es indispensable cuando se usa la función CERCA. Esto se puede ver en el listado 7 y en la figura #9!

La siguiente tabla nos muestra como varía DIS:

PARA CUADRO		PARA SUBCUADRO	
Dis	Longitud max. en Km.	Dis	Longitud max. en Km.
1	10 Km.	1	5 Km.
2	15 "	2	7.5 "
3	20 "	3	10 "
4	25 "	4	12.5 "

Como se ve en el listado 7, cuadros marcados con unos nos indican dónde nuestro predicado es verdadero, en la figura #9 de la carta Edafológica podemos comprobar el resultado obtenido y lo hacemos en la siguiente forma:

En el cuadro 21 hay caminos y al mismo tiempo hay una mina (X) dentro del mismo cuadro (un cuadro tiene 5Km. de lado). Vemos que hay minas a una distancia menor de 10Km. de cualquier tipo de camino. Por consiguiente, aparece un 1 en el cuadro 21.

En el cuadro 22 vemos que no existen minas, aún habiendo caminos, por lo que nuestro predicado se hace falso y aparece un cero.

Para el cuadro 29 vemos que existen 2 minas y en este cuadro no hay caminos, pero en sus vecinos si hay y están a menos de 10Km. de distancia, por lo tanto el predicado se hace verdadero y aparece un uno en el cuadro 29.

MINA

```
      LOGICAL FUNCTION MINA(N)
      LOGICAL MAYORQ,PRO
      EXTERNAL MAYORQ
      MINA=PRO(170.,MAYORQ,0)
      RETURN
      END
```

CAMINO

```
      LOGICAL FUNCTION CAMINO(N)
      LOGICAL MAYORQ,UNADE
      EXTERNAL MAYORQ
      CAMINO=UNADE(110,116,MAYORQ,0)
      RETURN
      END
```

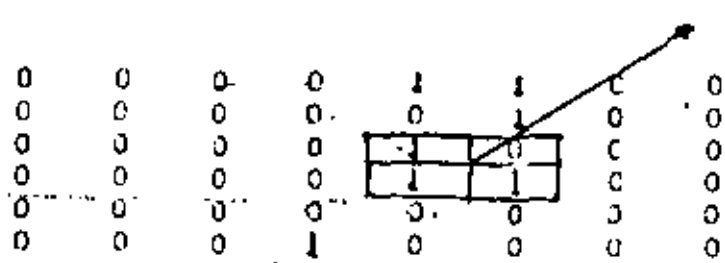
MIAL

```
      LOGICAL FUNCTION MIAL(N)
      LOGICAL CERCA
      INTEGER*2 MINAS(12,16),ICARRE(12,16)
      COMMON MINAS,ICARRE

C QUIERO SABER DONDE HAY UNA MINA QUE NO ESTÉ A MAS DE 10 KM. DE CUALQUIER TIP
C DE CAMINO
      MIAL=CERCA(MINAS,ICARRE,1,3)
      RETURN
      END
```

MAIN

```
      INTEGER*2 MINAS(12,16),ICARRE(12,16)
      LOGICAL MINA,CAMINO,MIAL
      EXTERNAL MINA,CAMINO,MIAL
      COMMON MINAS,ICARRE

      CALL IBUSCA(MINA,3,MINAS)
      CALL IBUSCA(CAMINO,3,ICARRE)
      CALL IBUSCA(MIAL,3,MINAS)
      STOP
      END
```

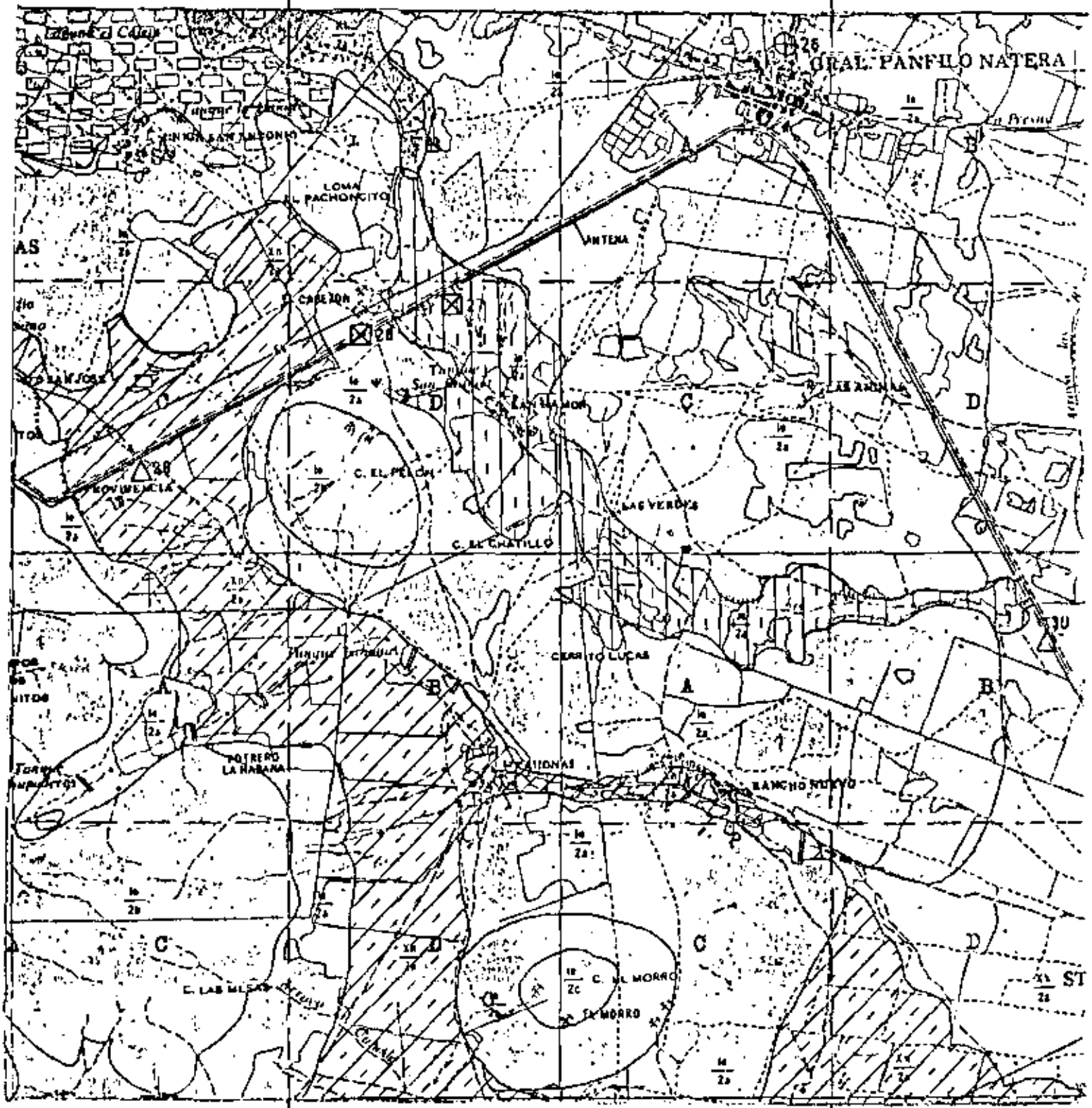
Equivalente a la Fig. 9



Listado 7

CUADRO #21

CUADRO #22



CUADRO #29

CUADRO #30

Fig. 9 Parte de la Carta Edafológica de Ojo Caliente, Zacatecas.

Para el cuadro 30 existen dos minas y caminos por lo que se cumple nuestro predicado y aparece un uno en el cuadro 30.

En el cuadro 21, la mina (⛏) aparece cerca de EL CABEZON. En los cuadros 29 y 30, las minas se localizan cerca del Cerro de EL MORRO, en los subcuadros 29d y 30c. (véase figura 9').

EJEMPLOS USANDO COMBINACIONES DE TODAS LAS FUNCIONES. 64

Con todas las funciones anteriormente definidas podemos hacer un gran uso de nuestro Banco de Datos Geográficos; los ejemplos mostrados en esta sección son interesantes porque poseen combinaciones de nuestras funciones lógicas, usando las expresiones lógicas .AND.,.OR. y .NOT. y sus combinaciones .AND..NOT. y .OR..NOT. Dentro de estos ejemplos se usan otras rutinas para funciones más específicas que complementan el uso del Banco de Datos Geográficos, rutinas que nos sirven para el diseño de carreteras (ver referencia 1), también rutinas que pueden determinar lugares no comunicados a ciertas distancias, etc.

A continuación analizaremos algunos de estos problemas con resultados obtenidos. Se hace notar en estos ejemplos que se pueden elaborar de muchas formas y obtener los mismos resultados. Hay algunos caminos más fáciles para solucionar un ejemplo, esto depende directamente de la habilidad y experiencia del usuario; tal vez varios usuarios resuelvan el mismo problema de diferentes formas (al decir formas, se refiere a la combinación diferente de las funciones estudiadas) y llegar a los mismos resultados.

EJEMPLO 1.

Se desea saber cuáles son los lugares que no están comunicados a una cierta distancia dada de vías de comunicación y sobre estos lugares diseñar una red de caminos (ref. 1). Para este caso se construyó una rutina que se llama BUSCAB, esta rutina trabaja igual que BUSCA, con la diferencia que hace algunos cálculos sobre las distancias a las vías de comunicación; en el listado 8 vemos el ejemplo con resultados :

1. Construímos nuestra función y le llamamos camino, esta función usa funciones lógicas anteriormente definidas en este caso UNADE y la función relacional MAYORQ. Como habíamos visto UNADE es un

.OR. y deseamos que nos busque todas las vías de comunicación comprendidas entre carretera de más de 2 carriles ó carretera pavimentada ó terracería transitable en todo tiempo ó terracería transitable en tiempo de secas ó carretera Federal ó carretera de cuota ó carretara estatal y la función queda:

```
LOGICAL FUNCTION CAMINO (N)
LOGICAL UNADE, MAYORQ
EXTERNAL MAYORQ
CAMINO = UNADE (110, 116, MAYORQ, 0)
RETURN
END
```

En el listado 8 vemos que en LOGICAL están también listados PRO, ENTRE, PROP, DIFERE, esto es debido a que como se corrían varios ejemplos por computadora en la tarjeta de LOGICAL se ponían casi todas las funciones para después solamente cambiar el resto de las tarjetas, ya que aunque se listen funciones que no se usan esto no afecta los resultados y la ejecución del programa.

2. El programa principal queda:

```
LOGICAL CAMINO
EXTERNAL CAMINO
CALL BUSCAB (CAMINO, 4)
STOP
END
```

Con esto hacemos que nos busque todos los tipos de caminos anteriormente mencionados a nivel de subcuadro y esto se muestra en el primer arreglo de ceros y unos. Sobre este arrego de ceros y unos BUSCAB trabaja sobre este arreglo y sobre la distancia deseada en este caso que es 2.5 Kms. El resulado está dado por un arreglo de E's y de carácter $ en las cuales E's son los subcuadros que están a más de 2.5 Kms. de la red de carreteras y caminos existentes en la zona, los subucadros marcados con el carácter $ son los subcuadros que no distan a más de 2.5 Kms. de la red de caminos y carreteras existentes. Sobre estas áreas marcadas de caracteres $'s se puede aplicar el algoritmo desarrollado en la referencia 1, el cual consiste en trazar una red de caminos en un área,

ningún punto debe quedar fuera de una distancia dada, en este caso sería la distancia de 2.5 Kms. En este ejemplo no se aplicó este algoritmo debido a que son áreas muy pequeñas; tal vez para el caso de que tengamos 4 cartas vaciadas en nuestro Banco podamos usar este algoritmo.

En la figura #10 que es una parte de la Carta Geológica de Ojo Caliente, Zac., vemos que los subcuadros c del cuadro 19 y a del cuadro 20 están marcados con líneas transversales y en el equivalente en el listado 8 vemos que tienen E's y esto indica que estos subcuadros distan a más de 2.5 Kms. de distancia de la red de caminos existente; en cambio los cuadros que no exceden la distancia de más de 2.5 Kms. de los caminos están etiquetados por $.

CAMINO DATE = 74193 08/52/

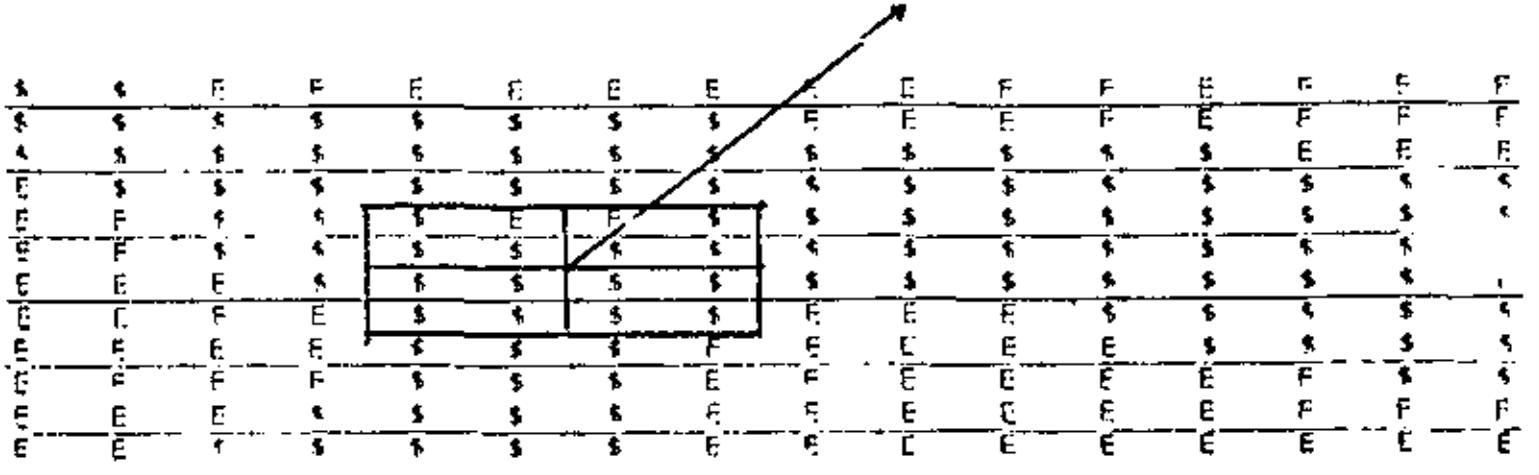
```
      LOGICAL FUNCTION CAMINO(N)
      LOGICAL ENTRE,PROP,DIFERE,PRO,UNADE,MAYORQ
      EXTERNAL ENTRE,DIFERE,MAYORQ
      CAMINO=UNADE(110,116,MAYORQ,0)
      RETURN
      END
```

MAIN DATE = 74198 08/52/

```
      LOGICAL CAMINO
      EXTERNAL CAMINO
C QUIERO SABER LOS SUBCUADROS QUE ESTAN A MAS DE 2.5 KM. DE LA RED DE
C CAMINOS Y CARRETERAS EXISTENTES
      CALL BUSCAB(CAMINO,4)
      STOP
      END
```

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	1	1	1	1	1	1	0	0
0	0	0	1	0	0	0	0	0	1	1	1	1	1	1	1
0	0	0	1	1	0	0	1	1	1	0	1	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	1	1	1	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0

Equivalente a la Fig. 10



CUADRO #19

CUADRO #20

CUADRO #27

CUADRO #28

Fig. 10 Parte de la Carta Geológica de Ojo Caliente, Zac. Hallazgo de cuadros incomunicados (más allá de 2.5 Km. de la red de caminos existente).

EJEMPLO 2

En este ejemplo se desea saber en qué lugares (por lugares se entiende cuadros o subcuadros) existe cuando menos una aeropista que no diste más de 15Kms. del pueblo llamado Gral. Pánfilo Natera y de los límites con San Luis Potosí. El interés de este ejemplo es poder usar nuestra función CERCA ya como una combinación lógica de funciones. También el interés radica en la combinación de otras funciones lógicas tales como PUEBLO, PRO, etc.

Considerando que las funciones utilizadas en este ejemplo ya fueron definidas anteriormente, la explicación para cada uno de los pasos de este ejemplo será breve.

1. Definimos nuestra función que nos servirá después para encontrar el pueblo que deseamos, en este caso usamos nuestra función llamada PUEBLO, como buscamos el pueblo de Gral. Pánfilo Natera, a nuestra función le ponemos un mnemónico llamado GRAL y queda de la siguiente forma:

```
      LOGICAL FUNCTION GRAL (N)
      LOGICAL PUEBLO
      GRAL = PUEBLO (2019)
      RETURN
      END
```

   El argurmento en la función pueblo es el #2019 que equivale al pueblo llamado GRAL. PANFILO NATERA y esto se puede ver en la tabla #4.

2. Definimos nuestra función aeropista y usamos nuestra función PRO, le llamamos avión y queda:

```
      LOGICAL FUNCTION AVION (N)
      LOGICAL MAYORQ, PRO
      EXTERNAL MAYORQ
      AVION = PRO (199., MAYORQ, 0)
      RETURN
      END
```

El 199 que aparece como un argumento en PRO es el equivalente numérico de aeropista y se ve en la tabla #1.

3. Definamos nuestra función que busca los límites con San Luis Potosí y le llamamos LIMITE, usamos PRO y queda como sigue:

```
LOGICAL FUNCTION LIMITE (N)
LOGICAL PRO, MAYORQ
EXTERNAL MAYORQ
LIMITE = PRO (150., MAYORQ, 0)
RETURN
END
```

El #150 que vemos en el primer argumento de PRO es el correspondiente numérico a límites con San Luis Potosí y se ve en la tabla #1.

4. Creamos la función importante que relaciona a las 3 funciones definidas anteriormente por medio de la función lógica CERCA y le llamamos APISTA y queda como sigue:

```
LOGICAL FUNCTION APISTA (N)
LOGICAL CERCA
INTEGER*2 IARRI (12,16), IARR2(12,16),IARR3(12,16)
COMMON IARRI, IARR2, IARR3
APISTA = CERCA (IARRI,IARR2 ,2,3).AND.CERCA (IARRI,IARR3 ,2,3)
RETURN
END
```

Vemos el valor de DIS (distancia) que es de 2, debido a que buscamos a no más de 15Kms. y el nivel es 3 debido a que buscamos a nivel de cuadro.

5. Creamos el programa principal en la forma siguiente:

```
INTEGER*2 IARRI(12,16),IARR2(12,16),IARR3(12,16)
LOGICAL GRAL, AVION, LIMITE, APISTA
EXTERNAL GRAL, AVION, LIMITE, APISTA
COMMON IARRI, IARR2, IARR3
CALL IBUSCA (AVION, 3, IARR1)
CALL IBUSCA (GRAL, 3, IARR2)
CALL IBUSCA (LIMITE, 3, IARR3)
CALL IBUSCA (APISTA, 3, IARR1)
STOP
END
```

En el listado 9 vemos que existe un 1, el cual nos indica que en este cuadro que es el número 14 nuestra función se hizo verdadera, en la Figura 11 que es una parte de la Carta Geológica de Ojo Caliente, Zac. vemos que efectivamente existe una aeropista (está marcada por un cuadro) y también observamos que no dista más de 15 Kms. del pueblo GRAL. PANFILO NATERA y de los límites con San Luis Potosí.

GRAL

```
      LOGICAL FUNCTION GRAL(N)
      LOGICAL PUEBLO
      GRAL=PUEBLO(2019)
      RETURN
      END
```

AVION

```
      LOGICAL FUNCTION AVION(N)
      LOGICAL MAYORQ,PRO
      EXTERNAL MAYORQ
      AVION=PRO(199.,MAYORQ,0)
      RETURN
      END
```

LIMITE

```
      LOGICAL FUNCTION LIMITE(N)
      LOGICAL PRO,MAYORQ
      EXTERNAL MAYORQ
      LIMITE=PRO(150.,MAYORQ,0)
      RETURN
      END
```

APISTA

```
      LOGICAL FUNCTION APISTA(N)
      LOGICAL CERCA
      INTEGER*2 IARR1(12,16),IARR2(12,16),IARR3(12,16)
      COMMON IARR1,IARR2,IARR3

C QUIERO UNA AEROPISTA QUE NO ESTE A MAS DE 15 KM. DE GRAL. PANFILO
C NATERA Y DE LOS LIMITES CON SAN LUIS POTOSI
      APISTA=CERCA(IARR1,IARR2,2,3).AND.CERCA(IARR1,IARR3,2,3)
      RETURN
      END
```

MAIN

```
      INTEGER*2 IARR1(12,16),IARR2(12,16),IARR3(12,16)
      LOGICAL GRAL,AVION,LIMITE,APISTA
      EXTERNAL GRAL,AVION,LIMITE,APISTA
      COMMON IARR1,IARR2,IARR3

      CALL IBUSCA(AVION,3,IARR1)
      CALL IBUSCA(GRAL,3,IARR2)
      CALL IBUSCA(LIMITE,3,IARR3)
      CALL IBUSCA(APISTA,3,IARR1)
      STOP
      END
```

```
0  0  0  0  0  0  0  0
0  0  0  0  0  1  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
```

Listado 9.

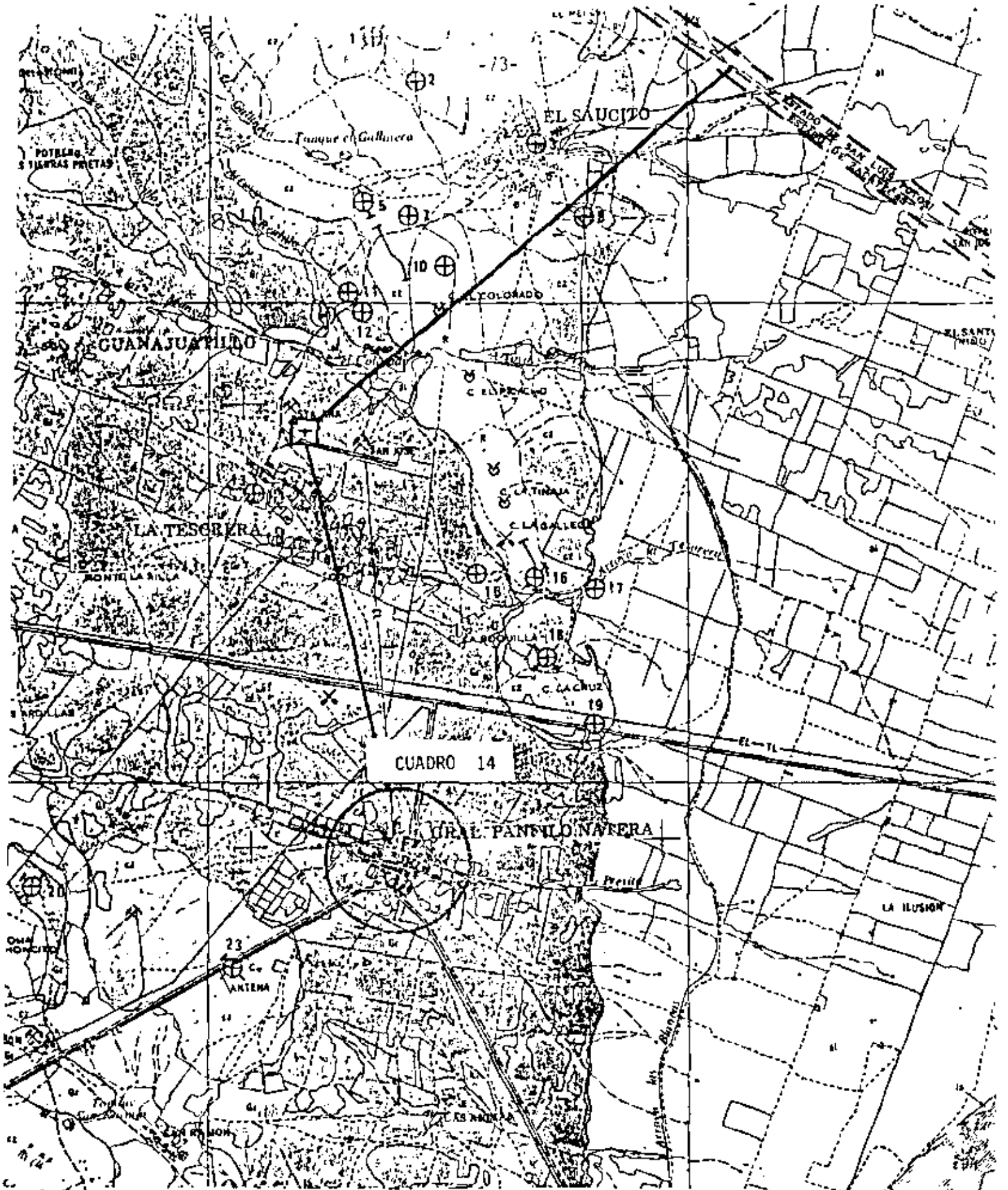


Fig. 11 Parte de la Carta Geológica de Ojo Caliente, Zacatecas

EJEMPLO 3

En este ejemplo vemos una gran aplicación del uso del Banco de Datos Geográficos para el diseño de carreteras a través de la función evalúa que nos da pesos para los subcuadros. En la primera parte del listado 10 vemos que se desea diseñar una carretera entre el pueblo llamado GRAL. PANFILO NATERA y el pueblo BAJIO DE SAN NICOLAS. En la figura 12 vemos que esa carretera ya existe, el propósito es probar el algoritmo desarrollado en la referencia 1 para diseño de carreteras (en la referencia 1 se usaban números aleatorios, en nuestro caso ya usamos pesos reales a las condiciones definidas). En el listado vemos los pesos que se toman para la función evalúa, éstos se describen en los comentarios del programa principal, en la segunda parte del listado 10 vemos los pesos asignados a cada cuadro según las propiedades. Es importante hacer notar el criterio para tomar los pesos y éstos son dados por la experiencia y conocimientos del usuario, los números marcados con el 90 son los subcuadros en donde se desarrolla el diseño de la carretera. La parte superior de la segunda parte del listado 10 marca las carreteras existentes en la zona y los pesos asignados a cada subcuadro por la función de evaluación llamada EVALUA; en el segundo arreglo numérico vemos la carretera marcada por los triángulos y su secuencia por números crecientes negativos; la Fig. 12 que es una parte de la carta geológica de Ojo Caliente, Zac. Tiene marcados los subcuadros por donde pasa la carretera existente, en la figura 13 tenemos marcados los subcuadros por los que pasa la carretera diseñada por la computadora. Como se puede observar la diferencia entre las figuras 12 y 13 es importante, ya que nos permite comparar.

El algoritmo del diseño de carreteras se muestra en la referencia 1

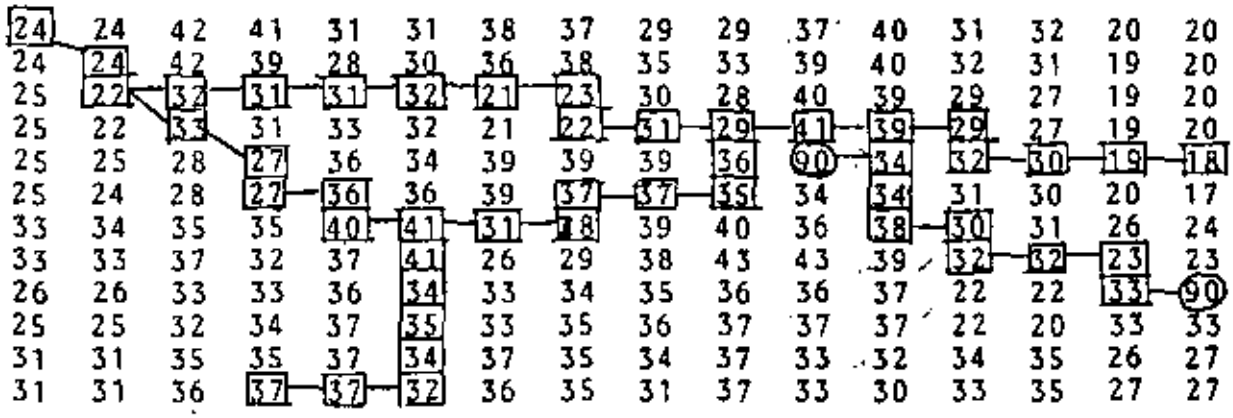
PRNECE

```
      LOGICAL FUNCTION PRNECE(N)
      PRNECE=.TRUE.
      RETURN
      END
```

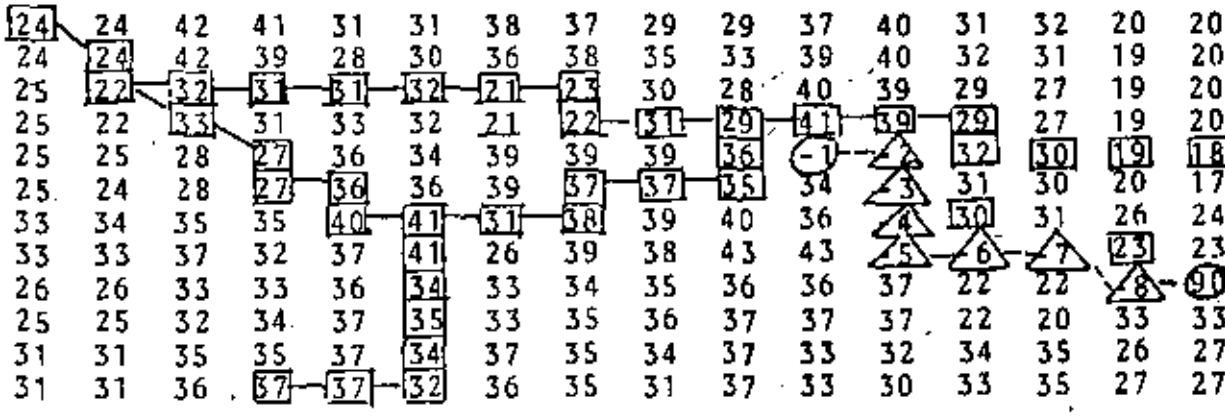
MAIN

```
      EXTERNAL PRNECE
      LOGICAL PRNECE
      INTEGER IAR(20,4)/128,170,181,184,185,186,210,211,800,801,802,803,
     1804,805,812,907,920,941,951,955,1,2,-3,-2,-4,-4,1,9,1,6,9,10,1,-1,
     29,-1,-1,-2,8,4,1,2,-3,-2,-4,-4,2,1,2,5,8,9,4,0,9,-1,0,-1,6,2,1,2,-
     33,-2,-4,-4,3,2,3,4,7,8,5,1,9,0,1,0,5,0/
C EVALUA LAS SIGUIENTES PROPIEDADES PARA EL DISEÑO DE CARRETERAS
C CON UN INTERVALO DE PESOS DE -10 AL0 Y SUS PESOS SON LOS SIGUIENTES
C DEPOSITO DE AGUA      128      1        1        1
C MINA                  170      2        2        2
C RIO                   181      -3       -3       -3
C ROCAS                 184      -2       -2       -2
C PANTANO               185      -4       -4       -4
C LAGUNA                186      -4       -4       -4
C PASTIZAL NATURAL      210      1        2        3
C PASTIZAL CULTIVADO    211      0        1        2
C TEXTURA GRUESA        800      1        2        3
C TEXTURA MEDIA         801      6        5        4
C TEXTURA FINA          802      9        8        7
C TERRENO PLANO         803      10       9        8
C TERRENO MONTUOSO      804      -1       4        5
C TERRENO MONTANOSO     805      -1       0        1
C NUMERO DE PUEBLOS     812      9        9        9
C ROCA VIOLETA          907      -1       1        0
C CALIZA                920      -1       0        1
C MARMOL                941      -2       -1       0
C SUELO ALUVIAL         951      8        6        5
C LITORAL               955      4        2        6
      CALL DEFINE(IAR,20,4)
      CALL EVALUA(4,PRNECE,1)
C QUIERO UNA CARRETERA ENTRE GRAL. PANFILO NATERA Y BAJIO DE SAN NICOLAS
      STOP
      END
```

Listado 10 (primera parte).



CARRETERAS EXISTENTES EN LA ZONA F-13-B-69



CARRETERA DISEÑADA

Listado 10 (segunda parte).

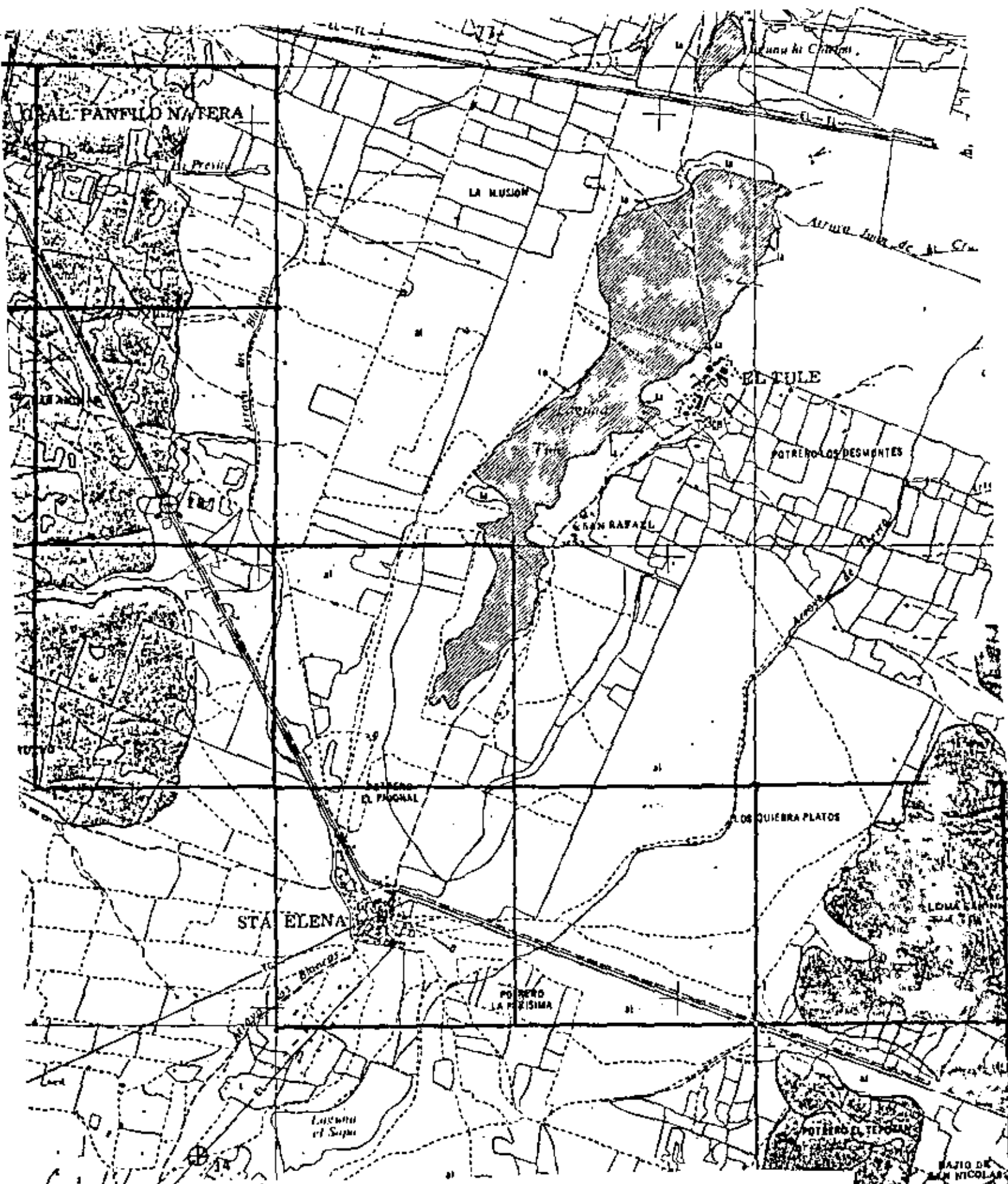


Fig. 12 Parte de la Carta Geológica de Ojo Caliente, Zacatecas.

los cuadros indican la carretera ya existente

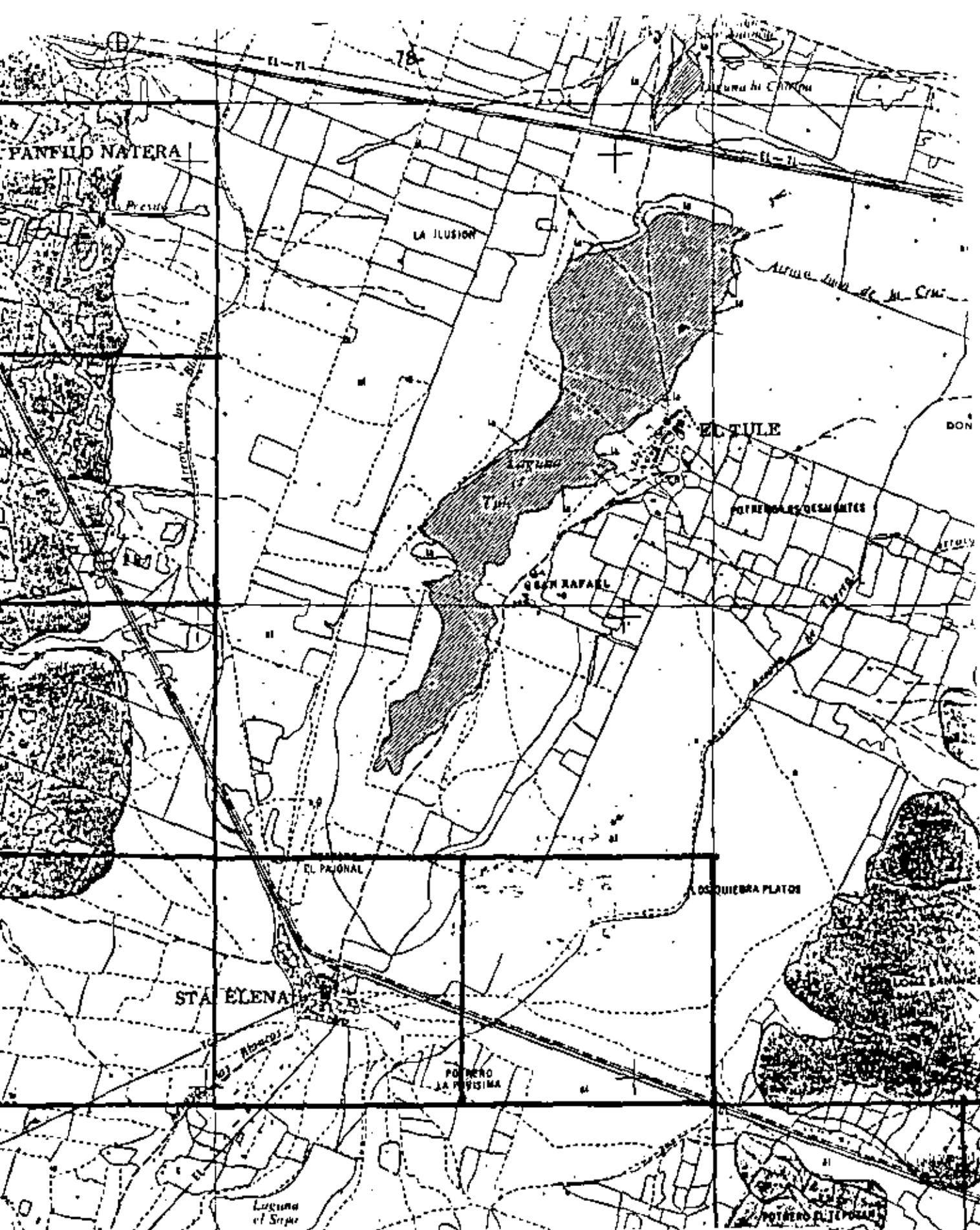


Fig. 13 Parte de la Carta Geológica de Ojó Caliente, Zacatecas.

los cuadros indican la carretera diseñadà por la computadora

EJEMPLO 4

En este ejemplo deseamos encontrar una región propia para el turismo. El propósito del desarrollo de este ejemplo es el de usar casi todas nuestras funciones y se muestra en los comentarios de la primera parte del listado 11 y el resultado se observa en la segunda parte del mismo listado en donde el cuadro 11 es el cuadro que cumple con todas nuestras condiciones. Las figuras 14 a la 18 nos muestran las propiedades de ese cuadro que hacen que nuestra función total sea verdadera.

```
LEVEL  21                REGION              DATE = 74129          11/26/28

      LOGICAL FUNCTION REGION(N)
      LOGICAL ENTRE,PROP,DIFERE,PRO,UNADE,MAYORQ,MENORQ,HAYVIA,SERPOB
     1,SERPRO
      EXTERNAL ENTRE,DIFERE,MAYORQ,MENORQ
      REGION=PROP(100.,ENTRE,50,60).AND.PRO(111.,MAYORQ,0).AND.PRO(121.,
     1MAYORQ,0).AND.PRO(123.,MAYORQ,0).AND.PRO(128.,MAYORQ,0).AND.PRO(13
     20.,MAYORQ,0).AND.UNADE(190,195,MENORQ,2300).AND.(.NOT.UNADE
     3(210,212,MAYORQ,0)).AND.UNADE(265,266,MAYORQ,20).AND.PRO(310.,MAY
     4ORQ,1000).AND.(.NOT.UNADE(300,302,MAYORQ,0)).AND.(.NOT.PRO(402.,
     5MAYORQ,0)).AND.UNADE(900,914,MAYORQ,0).AND.PRO(951.,MAYORQ,20).AND
     6.PRO(201.,MAYORQ,50).AND.PRO(269.,MENORQ,40).AND.PRO(296.,MAYORQ,1
     70).AND.PRO(304.,MAYORQ,0).AND.PRO(312.,MAYORQ,1).AND.(.NOT.UNADE(42
     867,430,MAYORQ,0)).AND.PRO(449.,MAYORQ,0).AND.PRO(674.,MAYORQ,15).AN
     90.PRO(794.,MAYORQ,50).AND.PRO(801.,MAYORQ,99).AND.PRO(812.,MAYORQ,
     100).AND.PRO(1166.,MAYORQ,0).AND.HAYVIA(11,2019).AND.HAYVIA(21,2019)
     2.AND.HAYVIA(23,2019).AND.SERPOB(3,256,0,0,0,0,0,0).AND.SERPOB(1,64
     3,1024,0,0,0,0,0).AND.SERPOB(2,1,0,0,0,0,0,0).AND.SERPOB(3,1,0,0,0,
     440,0,0).AND.SERPOB(7,4,0,0,0,0,0,0).AND.SERPRO(4,1).AND.SERPRO(3,0)
     5.AND.SERPRO(5,1).AND.SERPRO(7,1).AND.SERPRO(8,1)
C QUIERO UN CENTRO TURISTICO Y NECESITO UNA REGION QUE TENGA DEL 50 AL 60%
C DE CULTIVO, TENGA UNA CARRETERA PAVIMENTADA, LINEAS DE ENERGIA ELECTRICA
C LINEAS TELEFONICAS, DEPOSITOS DE AGUA, LAGUNAS, QUE ESTE A UN NIVEL MENOR
C DE 2300 METROS AL NIVEL DEL MAR, NO EXISTA PASTIZAL NATURAL, CULTIVADO
C O INDUCIDO, QUE TENGA ASOCIACIONES ESPECIALES DE VEGETACION TALES COMO
C  ZOTAL O NOPALERA ARRIBA DE UN 20%, QUE TENGA MAS DE 1000 HABITANTES, NO
C TENGA AGNAS INDUSTRIALES TALES COMO DE EXTRACCION DE PROCESAMIENTO O DE
C FABRICACION, QUE NO TENGA FACTORES LIMITANTES POR EL CLIMA EN SEGUNDO GRADO
C, Y QUE TENGA ROCAS IGNEAS YA SEA INTRUSIVA ACIDA O GRANITO O INTRUSIVA
C INTERMEDIA O DIORITA O INTRUSIVA BASICA O GRABO O EXTRUSIVA ACIDA O RIOLITA
C O EXTRUSIVA INTERMEDIA O ANDESITA O EXTRUSIVA BASICA O BASALTO O TOBA
C O BRECHA VOLCANICA O VITREA, QUE TENGA SUELO ALUVIAL MAYOR DEL 20%,
C QUE TENGA AGRICULTURA DE RIEGO TEMPORAL PERMANENTE ANUAL A MAS DEL 50%
C , QUE TENGA MATERIAL ESPINOSO MENOR DEL 40%, QUE TENGA CUERPOS DE AGUA
C ARTIFICIAL ESTACIONAL, QUE TENGA PUNTOS DE VERIFICACION, QUE EXISTA MAS
C DE UN PUEBLO EN LA REGION, QUE NO TENGA FACTORES LIMITANTES EN SEPTIMO
C GRADO DEL CLIMA O DE TOPOGRAFIA O DE EROSION O EXCESO DE AGUA, QUE
C TENGA UN CAMINO PROPUESTO DE OBRAS DE INFRAESTRUCTURA, QUE TENGA CAMBISOL
C EUTRICO MAYOR DEL 15% COMO DOMINANTE, QUE TENGA LITOSOL EUTRICO COMO
C DOMINANTE MAYOR DEL 50%, QUE TENGA TEXTURA MEDIA AL 100%, QUE TENGA
C TIPO PETROCALCICA Y QUE TENGA UN BANCO DE MATERIAL, QUE TENGA POBLADOS
C QUE TENGA CARRETERA PAVIMENTADA,LINEAS DE ENERGIA ELECTRICA Y TELEFONICAS A
C GRAN PANTLE NATERA, QUE TENGA DISTRIBUCION DE AGUA POR HUMANOS, TENGA ABASTEC
C AGUA POR MEDIO DE BORDO Y DE POZO, QUE TENGA MEDIOS DE ALMACENAMIENTO
C DE AGUA EN ALJIBE, QUE TENGA FORMA DE DISTRIBUCION DE AGUA POR MEDIO DE TUBERI
C , QUE TENGA ESCUELAS PRIMARIAS,QUE TENGA PROPUESTO UN TELEGRAFO,QUE NO TENGA
C ESCUELAS PROPUESTAS,QUE TENGA PROPUESTO DRENAJE POR FOSA SEPTICA,, QUE TENGA
C PROPUESTO UN CENTRO ASISTENCIAL,QUE TENGA PROPUESTAS LINEAS DE ENERGIA ELECTRI
      RETURN
      END
```

Listado 11 (primera parte).

```
      LOGICAL FUNCTION CAMINO(N)
      LOGICAL MAYORQ,UNADE
      EXTERNAL MAYORQ
      CAMINO=UNADE(110,116,MAYORQ,0)
      RETURN
      END

      LOGICAL FUNCTION TURIS(N)
      LOGICAL CERCA
      INTEGER*2 AREG(12,16),ACAM(12,16)
C QUIERO SABER DONDE HAY UNA REGION COMO LA MENCIONADA
C ANTERIORMENTE QUE NO ESTE A MAS DE 10 KM. DE CUALQUIER
C CAMINO
      TURIS=CERCA(AREG,ACAM,1,3)
      RETURN
      END

      INTEGER*2 AREG(12,16),ACAM(12,16)
      LOGICAL CAMINO,REGION,TURIS
      EXTERNAL CAMINO,REGION,TURIS
      CALL IBUSCA(REGION,3,AREG)
      CALL IBUSCA(CAMINO,3,ACAM)
      CALL IBUSCA(TURIS,3,AREG)
      STOP
      END
```

```
 0  0  0  0  0  0  0  0
 0  0  1  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0.
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
```

Listado 11 (segunda parte).

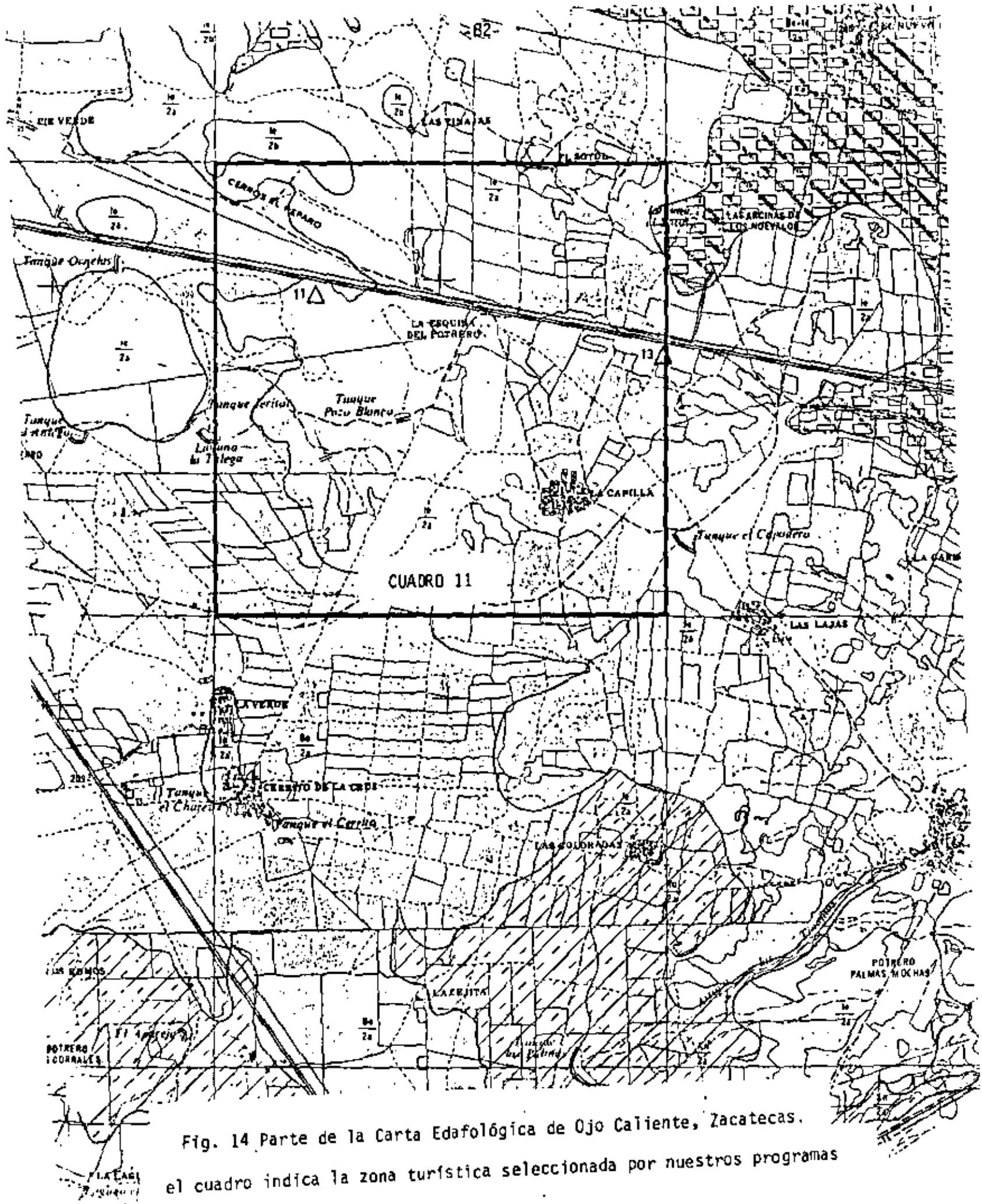


Fig. 14 Parte de la Carta Edafológica de Ojo Caliente, Zacatecas. el cuadro indica la zona turística seleccionada por nuestros programas

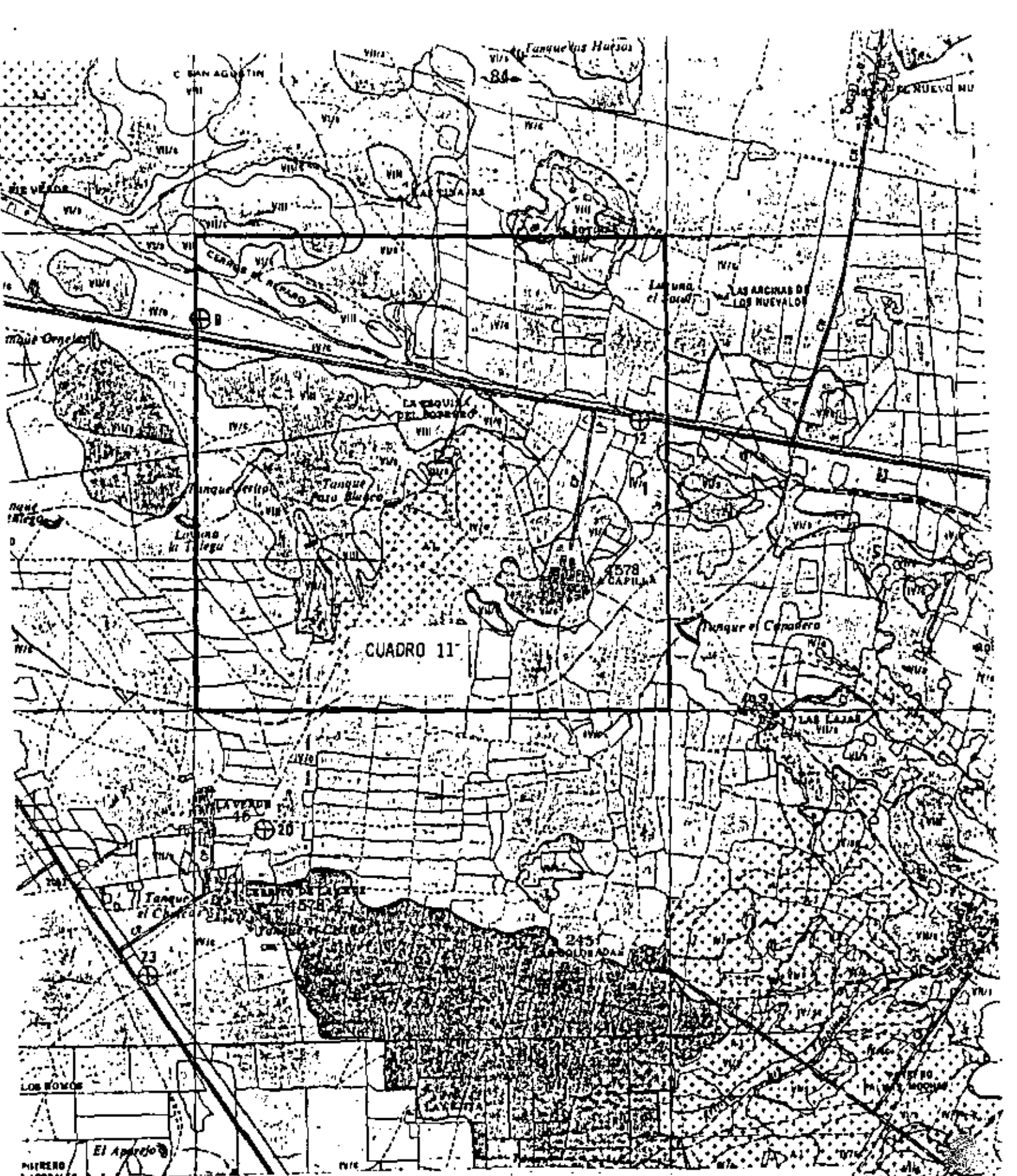


Fig. 16 Parte de la Carta de Uso Potencial de Ojo Caliente, Zacatecas.

el cuadro indica la zona turística seleccionada por nuestros programas

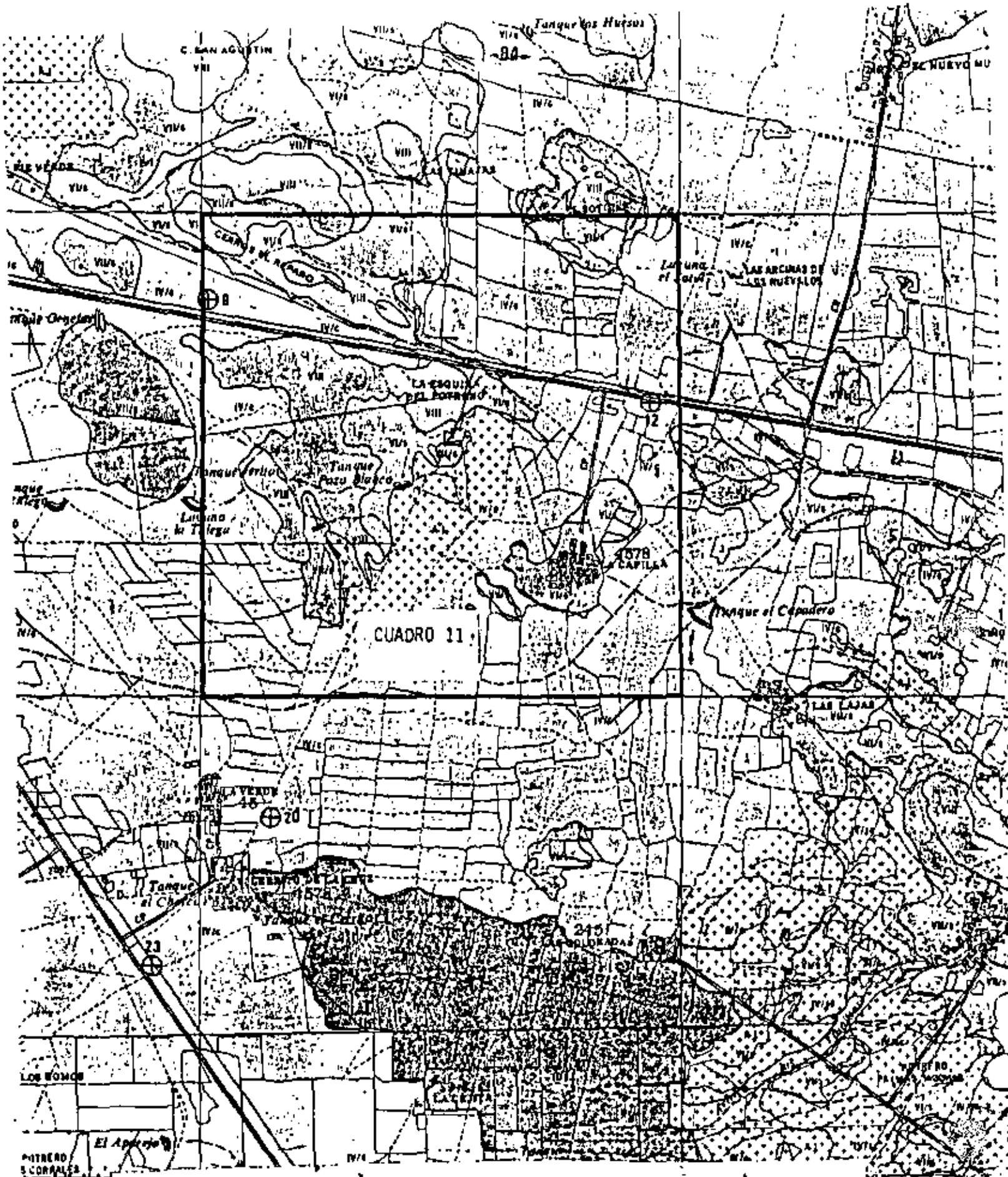


Fig. 16 Parte de la Carta de Uso Potencial de Ojo Caliente, Zacatecas.

el cuadro indica la zona turística seleccionada por nuestros programas

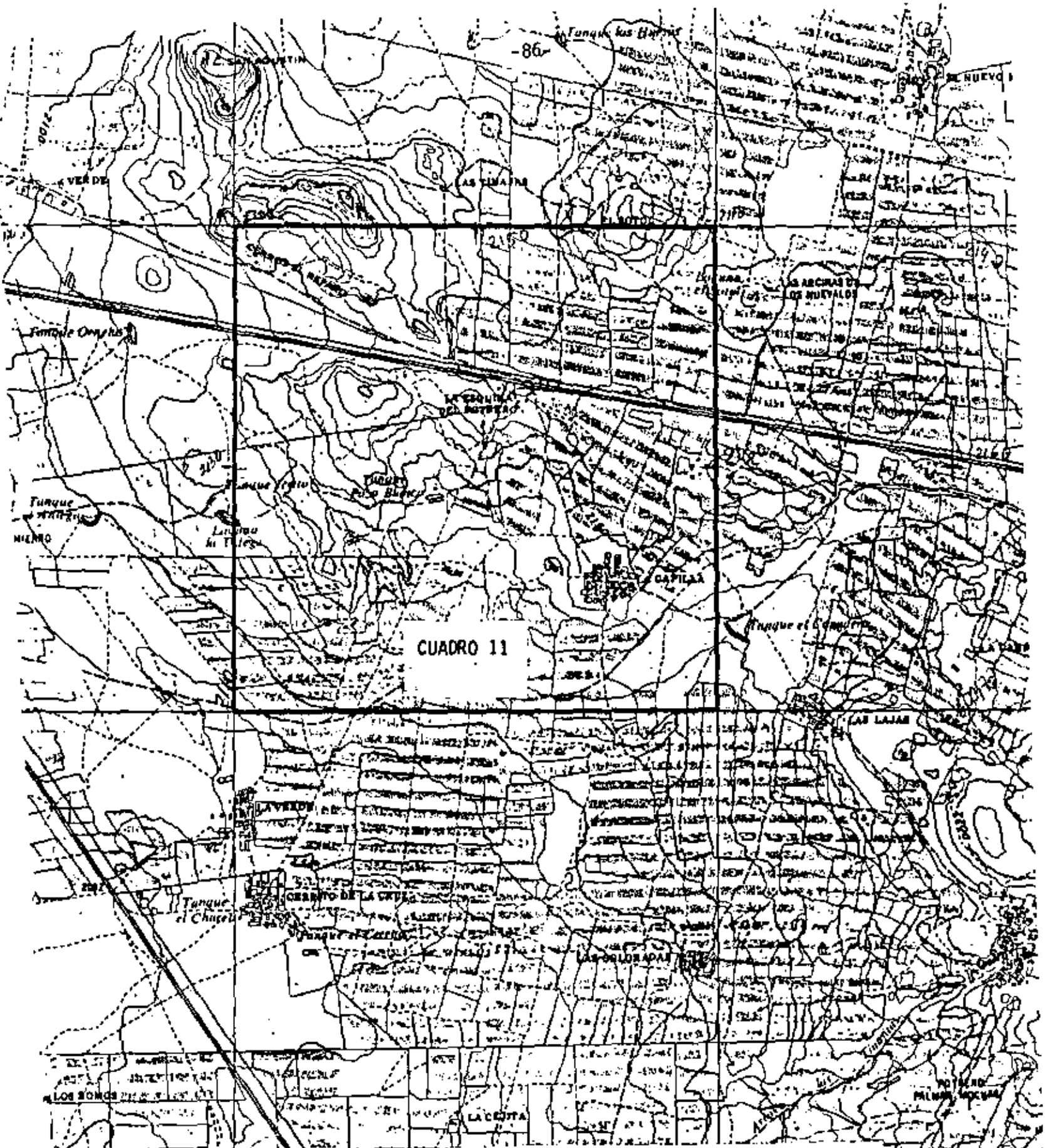


Fig. 18 Parte de la Carta Topográfica de Ojo Caliente, Zacatecas.

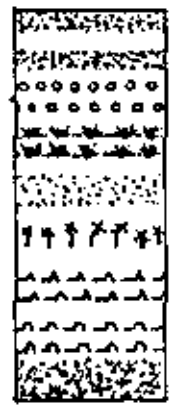
el cuadro indica la zona turística seleccionada por nuestros programas

TABLA # 1

APENDICE CARTA TOPOGRAFICA

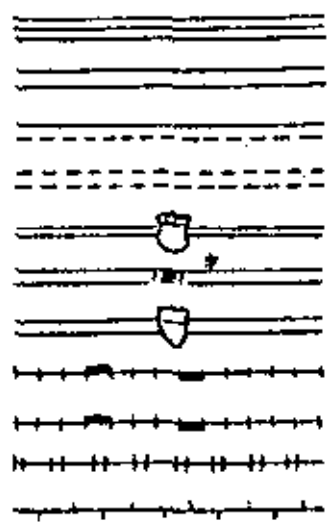
Todas las propiedades se expresan en % de terreno cubierto, a menos que se indique otra cosa

VEGETACION



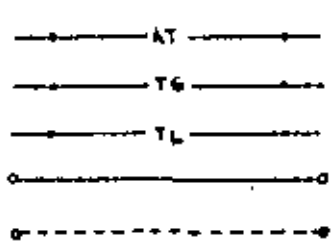
100. Cultivo
101 Huerto
102 Pastizal
103 Chaparral
104 Palmar
105 Bosque de Coníferas
106 Bosque de Latifoliadas
107 Manglar

CAMINOS Y FERROCARRILES (No. de ...)



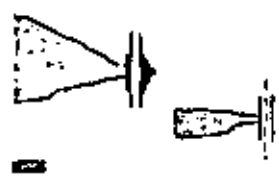
110 Carretera de más de dos Carriles
111 Carretera Pavimentada
112 Terracería Transitable en Todo Tiempo
113 Terracería Transitable en Tiempo de Secas
114 Carretera Federal
115 Carretera de Cuota
116 Carretera Estatal
117 Vía Sencilla de Ferrocarril
118 Estación de Ferrocarril
119 Vía Doble de FFCC
120 Otras Vías de FFCC

LINEAS DE CONDUCCION (No. de ...)



121 Líneas de Energía Eléctrica
122 Líneas de Telegrafo
123 Líneas Telefónicas
124 Líneas de Conducto Superficial
125 Líneas de Conducto Subterráneo

ALMACENAMIENTOS Superficial (No. de ...)



126 Presa
127 Bordo
128 Depósito de Agua

LIMITES Superficial (No. de ...)

130 Con Aguascalientes

150 Con San Luis Potosí

161 Con Zacatecas

162 Límites Estatales Verificados

163 Límites Estatales No Verificados

164 Límites Internacionales

PUNTOS DE CONTROL Superficial (No. de ...)

165 Vértice Geodésico

166 Apoyo Horizontal

167 Banco de Nivel de Precisión BN × 150

CULTURALES Superficial (No. de ...)

170 Mina

171 Faro

OROGRAFIA E HIDROGRAFIA Superficial (No. de ...)

180 Puente

181 Río

182 Cortina

183 Canal

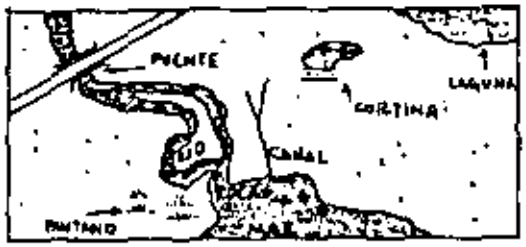
184 Rocas

185 Pantáno

186 Laguna

187 Lago

188 Mar



CURVAS DE NIVEL

190 Nivel en la Esquina Izquierda Superior

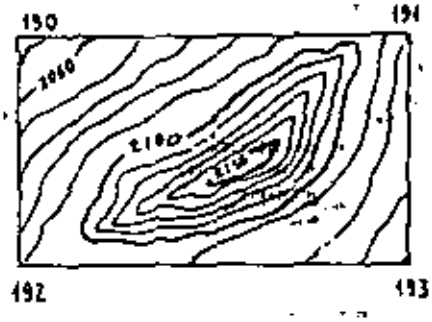
191 Nivel en la Esquina Derecha Superior

192 Nivel en la Esquina Izquierda Inferior

193 Nivel en la Esquina Derecha Inferior

194 Nivel Máximo

195 Nivel Mínimo



AEROPUERTOS Superficial (No. de ...)

196 Aeropuerto Internacional

197 Aeropuerto Local Pavimentado

198 Aeropuerto Local de Tierra

199 Aeropista

CARTA USO ACTUAL DEL SUELO

USO AGRICOLA		TIPO DE CULTIVO	
Ar	Agricultura de Riego	A	Anual
Atp	Agricultura de Temporal Permanente	P	Permanente
Atn	Agricultura de Temporal Nómada	Sp	Semipermanente

USO AGRICOLA- TIPO DE CULTIVO

200	Ar-A
201	Atp-A
202	Atn-A
203	Ar-P
204	Atp-P
205	Atn-P
206	Ar-Sp
207	Atp-Sp
208	Atn-Sp

USO PECUARIO

210	Pn	Pastizal Natural
211	Pc	Pastizal Cultivado
212	Pi	Pastizal Inducido

USO FORESTAL

220	FB	Bosque Natural
221	FBa	Bosque Artificial
222	FBg	Bosque de Galería
223	FBc	Bosque Caducifolio
224	C	Coníferas
225	L	Latifoliadas
231	(P)	Pino
232	(A)	Oyamel
233	(J)	Enebro
234	(Cu)	Cedro Blanco
235	(Q)	Encino
236	(Al)	Aile
237	(Li)	Liquidambar

238	(Po)	Alamo
239	(Sx)	Sáuce
240	(Eu)	Eucalipto
241	(Cs)	Casuarina
242	(Ma)	Paraíso
243	(Cm)	Cedro Rojo
244	(Rd)	Primavera
245	(Sm)	Pirul
250	FSa	Selva Alta
251	FSm	Selva Mediana
252	FSb	Selva Baja
253	(c)	Caducifolia
254	(p)	Perennifolia
255	(sc)	Sub-Caducifolia
256	(sp)	Sub-Perennifolia

ASOCIACIONES ESPECIALES DE VEGETACION

260	Pal	Palmar
261	Ma	Manglar
262	Po	Popal
263	Tu	Tular
264	Ca	Cardonal
265	Iz	Izotal
266	No	Nopalera
267	Sa	Sabána
268	Ch	Chaparral
269	Me	Matorral Espinoso
270	Mi	Matorral Inerme
271	Ms	Matorral Subinerme
272	S	Vegetación Secundaria.
273	H	Vegetación Halofita
274	Dc	Vegetación de Dunas Costeras
275	Da	Vegetación de Desiertos Arenosos
276	Pa	Vegetación de Páramos de Altura
277	Cr	Crasi-Rosulifolios Espinosos
278	Mz	Mezquital

279	Qt	Encinar Tropical
280	G	Vegetación de galería

DESPROVISTO DE VEGETACION

281	Des	Areas en Proceso de Desmonte
282	EoF	Erosión Eólica Fuerte
283	EoM	Erosión Eólica Moderada
284	EoL	Erosión Eólica Leve
285	EhF	Erosión Hídrica Fuerte
286	EhM	Erosión Hídrica Moderada
287	EhL	Erosión Hídrica Leve
288	Er	Eriales
289	Do	Dunas Costeras
290	Dr	Desiertos Arenosos
291	Sl	Salinas
292	Sc	Escoria

CUERPOS DE AGUA Propiedades Superficiales:

293	np	Natural Permanente	
294	ne	Natural Estacional	
295	ap	Artificial Permanente	X 10
296	ae	Artificial Estacional	X 10
297	⋊⊂	Piscicultura	X 10

ZONAS INDUSTRIALES

300	ZI(E)	De Extracción
301	ZI(P)	De Procesamiento
302	ZI(F)	De Fabricación
303	☼	Aserradero Superficial (No. de ...)
304	⊕	Punto de Verificación Superficial (No. de ...)

SERVICIOS EN LA POBLACION Superficial:

310	Número de Habitantes (en poblaciones)
311	Número de Ciudades (Ciudad es mayor de 50,000 habitantes)
312	Número de Pueblos

(Otros Servicios en la Población se Registran en Categorías Puntuales)

CARTA DE USO POTENCIAL

USO DEL SUELO
I
II
III
IV
V
VI
VII
VIII

FACTORES LIMITANTES	
s	Suelo
c	Clima
t	Topografía
e	Erosión
i	Exceso de Agua

400	I								
401	IIs	402	IIc	403	IIt	404	IIe	405	IIi
406	IIIs	407	IIIc	408	IIIt	409	IIIe	410	IIIi
411	IVs	412	IVc	413	IVt	414	IVe	415	IVi
416	Vs	417	Vc	418	Vt	419	Ve	420	Vi
421	VIs	422	VIc	423	VIt	424	VIe	425	VIi
426	VIIs	427	VIIc	428	VIIt	429	VIIe	430	VIIi
431	VIIIs	432	VIIIc	433	VIIIt	434	VIIIe	435	VIIIi

PROPOSICIONES

OBRAS DE INFRAESTRUCTURA Superficial (No. de ...)

440	BI	Boquilla de Irrigación
441	BR	Boquilla para Control de Avenidas
442	BP	Boquillas para Piscicultura
443	BA	Boquillas para Abastecimiento de Agua
444	BE	Boquillas para Generación de Energía
445	BZ	Boquillas para Control de Azolves
446	AP	Aeropistas
447	OC	Obras de Captación
448	OD	Obras de Defensa
449	CP	Camino Propuesto

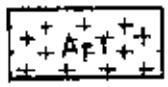
450	CR	Camino por Reconstruir
451	PU	Puertos
452	Ps	Piscicultura
453	⊗	Punto de Verificación

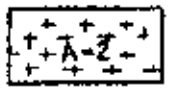
SERVICIOS PARA LA POBLACION (Proposiciones)

(Estos se registran como información puntual)

CONTROL DE EROSION

460	A1	Area que Requiere Control Inmediato
461	A2	Area que Requiere Control Futuro





CARTA EDAFOLOGICA

UNIDADES DE SUELO DOMINANTE	SECUNDARIO		
500	501	J	Fluvisol
502	503	Jd	Fluvisol Districo
504	505	Je	Fluvisol Eútrico
506	507	Jk	Fluvisol Calcárico
508	509	Jg	Fluvisol Gléyico
510	511	R	Regosol
512	513	Rd	Regosol Districo
514	515	Re	Regosol Eútrico
516	517	Rk	Regosol Calcárico
520	521	Q	Arenosol
522	523	Qd	Arenosol Districo
524	525	Qe	Arenosol Eútrico
530	531	G	Gleysol
532	533	Gn	Gleysol Háplico
534	535	Gh	Gleysol Húmico
536	537	Gk	Gleysol Cálcico
538	539	Gd	Gleysol Thionico
540	541	Gp	Gleysol Plíntico
542	543	Gm	Gleysol Hístico
550	551	E	Rendzina
560	561	U	Ranker
570	571	T	Andosol
572	573	Tn	Andosol Háplico
574	575	Tv	Andosol Vítrico
576	577	Tg	Andosol Gléyico
580	581	V	Vertisol
590	591	Y	Yermosol
592	593	Yn	Yermosol Háplico
594	595	Yk	Yermosol Cálcico
596	597	Yy	Yermosol Gypsico
598	599	Yl	Yermosol Lúvico.

DOMINANTE	SECUNDARIO		
600	601	X	Xerosol
602	603	Xn	Xerosol Háplico
604	605	Xk	Xerosol Cálcico
606	607	Xy	Xerosol Gypsico
608	609	Xl	Xerosol Lúvico
610	611	Z	Solonchak
612	613	Zn	Solonchak Háplico
614	615	Zh	Solonchak Húmico
616	617	Zt	Solonchak Takyrico
618	619	Zg	Solonchak Gléyico
620	621	S	Solonetz
622	623	Sn	Solonetz Háplico
624	625	Sh	Solonetz Húmico
626	627	Sg	Solonetz Gléyico
630	631	W	Planosol
632	633	Wn	Planosol Háplico
634	635	Wh	Planosol Húmico
636	637	Ws	Planosol Solódico
640	641	K	Castañosem
642	643	Kn	Castañosem Háplico
644	645	Kk	Castañosem Cálcico
646	647	Kl	Castañosem Lúvico
650	651	C	Chernozem
652	653	Cn	Chernozem Háplico
654	655	Ck	Chernozem Cálcico
656	657	Cl	Chernozem Lúvico
658	659	Cg	Chernozem Gléyico
660	661	H	Phaeozem
662	663	Hn	Phaeozem Háplico
664	665	Hk	Phaeozem Calcárico
666	667	Hl	Phaeozem Lúvico
668	669	Hg	Phaeozem Gléyico

DOMINANTE	SECUNDARIO		
670	671	B	Cambisol
672	673	Bn	Cambisol Háplico
674	675	Be	Cambisol Eutrico
676	677	Bk	Cambisol Calcárico
678	679	Bv	Cambisol Vértico
680	681	Bh	Cambisol Húmico
682	683	Bt	Cambisol Andico
690	691	L	Luvisol
692	693	Ln	Luvisol Háplico
694	695	Lc	Luvisol Crómico
696	697	Lf	Luvisol Férrico
698	699	La	Luvisol Albico
700	701	Lp	Luvisol Plintico
702	703	Lg	Luvisol Gléyico
710	711	D	Podzoluvisol
712	713	Dn	Podzoluvisol Háplico
714	715	Dg	Podzoluvisol Gléyico
720	721	P	Podzol
722	723	Pn	Podzol Humo-férrico
724	725	Pf	Podzol férrico
726	727	Ph	Podzol Húmico
728	729	Po	Podzol Ocrico
730	731	Pi	Podzol Plácico
732	733	Pg	Podzol Gléyico
740	741	A	Acrisol
742	743	An	Acrisol Háplico
744	745	Ah	Acrisol Húmico
746	747	Ap	Acrisol Plíntico
748	749	Ag	Acrisol Gléyico
750	751	N	Nitosol
752	753	Nd	Nitosol Dístrico
754	755	Ne	Nitosol Eutrico

DOMINANTE	SECUNDARIO		
760	761	F	Ferralsol
762	763	Fn	Ferralsol Háplico
764	765	Fo	Ferralsol Ocrico
766	767	Fr	Ferralsol Rodico
768	769	Fh	Ferralsol Húmico
770	771	Fp	Ferralsol Plíntico
780	781	M	Histosol
782	783	Md	Histosol Dístrico
784	785	Me	Histosol Eutrico
790	791	I	Litosol
792	793	Id	Litosol Dístrico
794	795	Ie	Litosol Eutrico

CLASE TEXTURAL

800	1	Textura Gruesa
801	2	Textura Media
802	3	Textura Fina

CLASE TOPOGRAFICA (Pendiente)

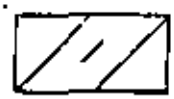
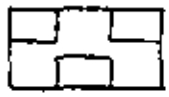
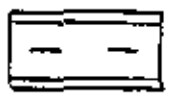
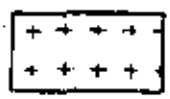
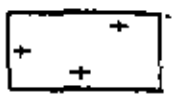
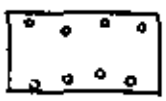
803	a	Terreno Plano a Ligeramente Ondulado- Pendientes Menores de 8%
804	b	De Lomerío a Terreno Montuoso- Pendientes entre 8 y 20%
805	c	De Terreno con Disección Severa a Terreno Montañoso- Pendientes Mayores de 20%.

FASES

SALINA— Expresada como Conductividad Eléctrica del Extracto de Saturación de por lo menos una Parte del Suelo a menos de 125cm. de Profundidad Medida en mmhos./cm. a 25°C.

806	ls	Suelo Ligeramente Salino. Conductividad de 4 a 8 mmhos./cm.
807	ms	Suelo Moderadamente Salino. Conductividad de 9 a 15 mmhos./cm.
808	fs	Suelo Fuertemente Salino. Conductividad de 16 ó más mmhos./cm.
809	n	SODICA— Suelos con Más del 15% de Saturación de Sodio en Alguna Porción a menos de 125cm. de Profundidad. No se Usa en Solonetz.

TIPOS

810		Dúrica
811		Dúrica Profunda
812		Petrocálcica
813		Petrocálcica Profunda
814		Frágica
815		Concrecionaria
816		Lítica
817		Lítica Profunda
818		Gravosa
819		Pedregosa

TIPOS

810 Dúrica (Duripan a menos de 50cm. de Profundidad)

811 Dúrica Profunda (Duripan entre 50 y 100 cm. de Profundidad)

812 Petrocálcica (Horizonte Petrocálcico a menos de 50 cm. de Profundidad)

813 Petrocálcica Profunda (Horizonte Petrocálcico entre 50 y 100 cm. de Profundidad)

814 Frágica (Fragipan a menos de 100 cm. de Profundidad)

815 Concrecionaria (Horizonte Concrecionario a menos de 100 cm. de Profundidad)

816 Lítica (Lecho Rocoso entre 25 y 50 cm. de Profundidad)

817 Lítica Profunda (Lecho Rocoso entre 50 t 100 cm. de Profundidad)

818 Gravosa (Fragmentos menores de 7.5 cm. en la Superficie ó cerca de ella, que Impiden el Uso de Máquina Agrícola)

819 Pedregosa (Fragmentos mayores de 7.5 cm. en la Superficie ó cerca de ella, que Impiden el Uso de Máquina Agrícola)

PUNTOS DE VERIFICACION Superficial (No. de ...)

820 ⊕ Sín Muestreo

821 △ Con Muestreo Superficial

822 ⊠ Pozo a Cielo Abierto

CARTA GEOLOGICA

ROCAS IGNEAS

900	Igia	Intrusiva Acida
901	Gr	Granito
902	Igii	Intrusiva Intermedia
903	D	Diorita
904	Igib	Intrusiva Básica
905	Ga	Gabro
906	Igea	Extrusiva Acida
907	R	Riolita
908	Igei	Extrusiva Intermedia
909	A	Andesita
910	Igeb	Extrusiva Básica
911	B	Basalto
912	T	Toba
913	Br	Brecha Volcánica
914	V	Vítrea

ROCAS SEDIMENTARIAS

920	cz	Caliza
921	cz-lu	Caliza-Lutita
922	ma	Marga
923	lu	Lutita
924	lu-ar	Lutita-Arenisca
925	ar	Arenisca
926	ar-cg	Arenisca-Conglomerado
927	cg	Conglomerado
928	br	Brecha
929	y	Yeso
930	tr	Travertino
931	ti	Tilita

ROCAS METAMORFICAS

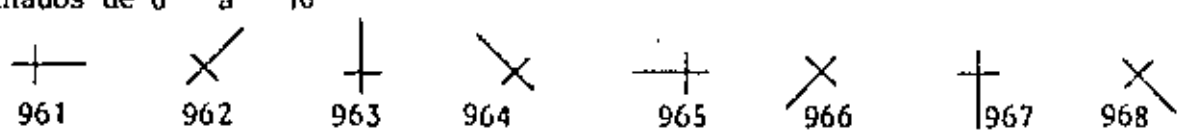
940	C	Cuarcita
941	M	Mármol
942	P	Pizarra
943	E	Esquisto
944	Gn	Gneis

SUELOS

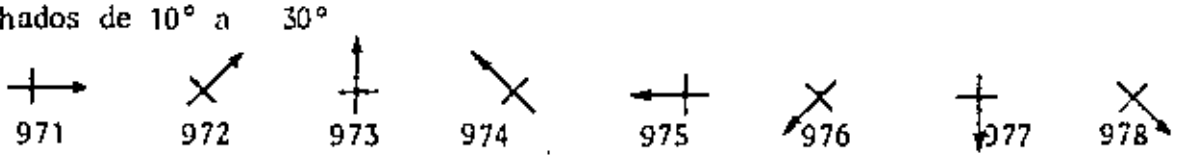
950	re	Residual
951	al	Aluvial
952	la	Lacustre
953	pi	Piamonte
954	pa	Palustre
955	li	Litoral
956	eo	Eólico
957	gl	Glacial

ESTRUCTURAS Superficial (No. de ...)

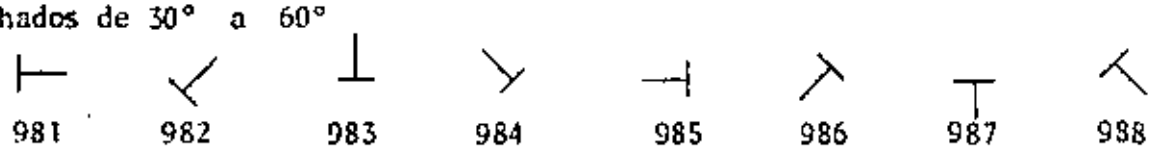
Echados de 0° a 10°



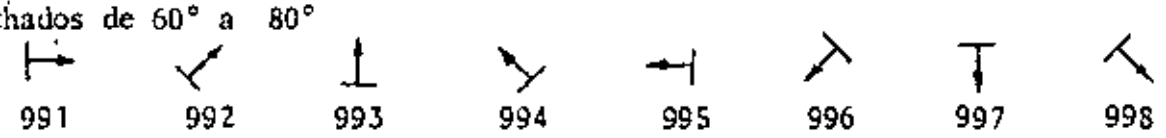
Echados de 10° a 30°



Echados de 30° a 60°



Echados de 60° a 80°



Echados de 80° a 90°

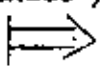

1001


1002


1003


1004

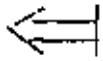
Echado y Rumbo de Flujos de Rocas Igneas

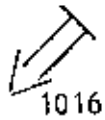

1011

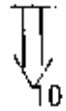

1012


1013


1014


1015


1016


1017


1018

Rumbo y Echado de Foliación


1021


1022


1023


1024

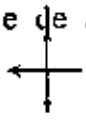

1025


1026


1027


1028

Eje de Anticlinal


1031


1032


1033


1034


1035


1036


1037


1038

Eje de Anticlinal Recumbente


1041


1042


1043


1044


1045


1046


1047


1048

Domo


1051


1052

Eje de Sinclinal


1061


1062

1063

1064

Eje de Sinclinal Recumbente

1071

1072

1073

1074

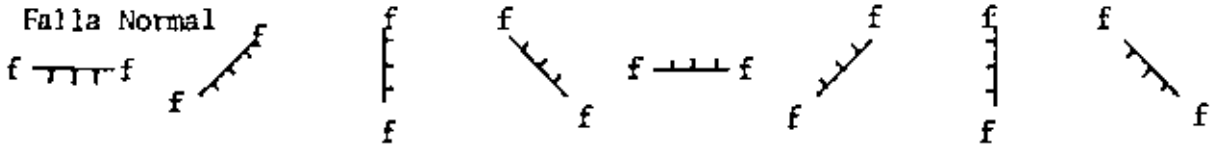
1075

1076

1077

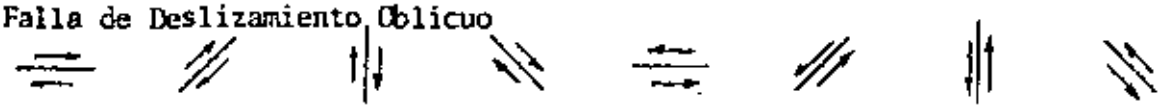
1078

Falla Normal



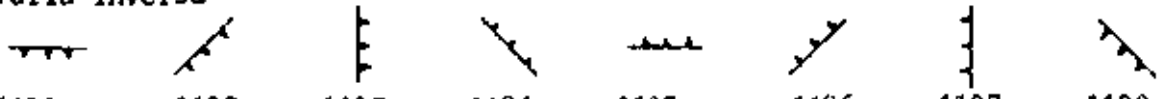
1081 1082 1083 1084 1085 1086 1087 1088

Falla de Deslizamiento Oblicuo



1091 1092 1093 1094 1095 1096 1097 1098

Falla Inversa



1101 1102 1103 1104 1105 1106 1107 1108

Fractura

1111 1112 1113 1114

Dique

1121 1122 1123 1124

Veta

1131 1132 1133 1134

Contacto

1141 1142 1143 1144

Contacto Inferido

1151 1152 1153 1154

Superficial (No. de ...)

1160	Volcán	
1161	Dolina	
1162	Manantial Frío	
1163	Manantial Termal	
1164	Mina (también se registran en las propiedades 170-175)	
1165	Cata	
1166	Banco de Material	

TABLA # 2

PROPOSICIONES

OBRAS DE INFRAESTRUCTURA

BI	BOQUILLA DE IRRIGACION
BE	BOQUILLAS PARA CONTROL DE AVENIDAS
BP	BOQUILLAS PARA PISCICULTURA
BA	BOQUILLAS PARA ABASTECIMIENTO DE AGUA
BE	BOQUILLA PARA GENERACION DE ENERGIA
BZ	BOQUILLA PARA CONTROL DE AZOLVES
AP	AEROPISTAS
OC	OBRAS DE CAPTACION
OD	OBRAS DE DEFENSA
CP	CAMINO PROPUESTO
CR	CAMINO POR RECONSTRUIR
PU	PUERTOS
PS	PISCICULTURA
⊕	PUNTO DE VERIFICACION

SERVICIOS PARA LA POBLACION

1	ABASTECIMIENTO DE AGUA DE FUENTE SUPERFICIAL
2	ABASTECIMIENTO DE AGUA DE FUENTE SUBTERRANEA
3	ESCUELA
4	TELEGRAFO
5	DRENAJE POR FOSA SEPTICA
6	DRENAJE POR EMISOR
7	CENTRO ASISTENCIAL
8	ENERGIA ELECTRICA

CONTROL DE EROSION

A-1	AREA QUE REQUIERE CONTROL INMEDIATO
A-2	AREA QUE REQUIERE CONTROL AL FUTURO

TABLA # 3

SERVICIOS EN LA POBLACION

1 ABASTECIMIENTO DE AGUA
- a MANANTIAL
- b RIO
- c LAGO O LAGUNA
- d BORDO
- e PRESA
- f POZO
- g POZA

2 MEDIO DE ALMACENAMIENTO
- a ALIBE
- b TANQUE ELEVADO
- c CAJA DE AGUA

3 FORMA DE DISTRIBUCION
- a TUBERIA
- b CANAL
- c VEHICULO MOTORIZADO
- d TRACCION ANIMAL
- e HUMANO

4 DRENAJE
- a EMISOR
- b FOSA SEPTICA
- c FOSA

5 ASISTENCIAL
- a HOSPITAL
- b CLINICA

6 MUNICIPAL
- a RASTRO
- b CEMENTERIO

7 EDUCACIONAL
- a PRE PRIMARIA
- b PRIMARIA
- c SECUNDARIA
- d PREPARATORIA
- e NORMAL
- f ENSEÑANZA TECNICA
- g ENSEÑANZA SUPERIOR

8 CORRIENTE ELECTRICA
- a POR LINEA
- b PLANTA PROPIA

9 COMUNICACIONES
- a CORREO
- b TELEGRAFO
- c TELEFONO
- d RADIO COMUNICACION
- e RADIO DIFUSORA

|CLAVE| NO ESTA EN SERVICIO

EJEMPLO DOLORES 625 1b-2c-3a-4c-5b-6ab (2)-7b (3/5)-8a-9a | b |

SIGNIFICADO POBLACION DOLORES

625	HABITANTES
1b	ABASTECIMIENTO DE AGUA DE RIO
2c	ALMACENAMIENTO EN CAJA
3a	DISTRIBUCION POR TUBERIA
4c	DRENAJE EN POSA
5b	EXISTE UNA CLINICA
6ab (2)	CUENTA CON UN RASTRO Y DOS CEMENTERIOS
7b(3/5)	EXISTE UNA PRIMARIA HASTA TERCER AÑO
8a	CUENTA CON CORRIENTE ELECTRICA POR LINEA
9a | b |	EXISTE UNA OFICINA DE CORREOS Y UNA DE TELEGRAFOS QUE NO OPERA

TABLA # 4

NUM. PUEBLO	NOMBRE DEL PUEBLO
2001	PIE VERDE
2002	EL NUEVO MUNDO
2003	EL SAUCITO
2004	LA CANDELARIA
2005	EL CARMEN
2006	LA ESQUINA DEL POTRERO
2007	LA CAPILLA
2008	GUANAJUATILLO
2009	LA TESORERA
2010	SAN JOSE EL SALADILLO
2011	LA VERDE
2012	CERRITO DE LA CRUZ
2013	LA CUJITA
2014	LAS COLORADAS
2015	LAS LAJAS
2016	POZO JARILLAS
2017	UNION SAN ANTONIO
2018	SAN RAMON
2019	GRAL. PANFILO NATERA
2020	LAS VERDES
2021	EL TULE
2022	EL REFUGIO
2023	DOLORES
2024	SAN CRISTOBAL
2025	TAHONAS
2026	RANCHO NUEVO
2027	STA. ELENA
2028	ESTACION BERRIOZABAL
2029	LA PALMA
2030	OJO CALIENTE

NUM. PUEBLO	NOMBRE DEL PUEBLO
2031	PAPANTON VIEJO
2032	JARRILLAS
2033	LA HACIENDITA
2034	STA. MA. LA PAZ
2035	SAN BLAS COPUDAS
2036	EL TILDIO
2037	SAN PABLO
2038	BAJIO DE SAN NICOLAS
2039	PIEDRA GORDA
2040	COL. 20 DE NOVIEMBRE
2041	LA CONCEPCION
2042	HIDALGO
2043	MILAGROS
2044	SAN JOSE DE LOS LLANOS
2045	STO. TOMAS VENADITOS
2046	CERRITOS DE AGUA

TABLA # 5

TIPO DE CAMINO O CONDUCTO		
1	Camino propuesto a	CP
2	Camino para reconstruirse a	CR
10	Carretera de más de 2 caminos a	
11	Carretera pavimentada a	
12	Terracería transitable en todo tiempo a	
13	Terracería transitable sólo en tiempo de secas a	
14	Carretera federal a	
15	Carretera de cuota a	$
16	Carretera estatal a	
17	FFCC Vía sencilla a	
19	FFCC Vía doble a	
20	Otras vías FFCC a	
21	Línea de energía eléctrica a	AT
22	Línea telegráfica a	TG
23	Línea telefónica a	TL
24	Conducto superficial a	

AGRADECIMIENTOS

A la COMISION DE ESTUDIOS DEL TERRITORIO NACIONAL por la reproducción de una porción de sus mapas obtenidos. Los mapas completos se pueden obtener en San Antonio Abad No. 124 México 8 D. F.

A las autoridades de IBM de México, S.A., en especial a su Presidente, Sr. Gustavo De La Serna Valdivia, por su apoyo constante y decidido.

REFERENCIAS

1.- Bribiesca, E., y Avilés, R.
"Codificación en cadenas y técnicas de reducción de información para mapas y dibujos lineales".
Reporte CCAL-74-7 Centro Científico IBM de América Latina. Agosto, 1974. (México)

2.- A. Guzmán y E. Bribiesca.
"Multiple-level geographic data bank for direct access storage"
Reporte CCAL-74-9 Centro Científico IBM de América Latina" 1974. (México)

3.- Guzmán A. y Bribiesca E.
"Utilización de un Banco de Datos Geográficos"
Memorias del I CONGRESO PANAMERICANO Y III NACIONAL DE FOTOGRAMETRIA, FOTOINTERPRETACION Y GEODESIA. Julio, 1974. (México)

4.- Alfonso Torres Roqueñi y José Oliveres Vidal.
"Localización por computadora de cuencas lecheras"
Ponencia presentada en el SEGUNDO CONGRESO INTERAMERICANO DE SISTEMAS E INFORMATICA. Noviembre, 1974. (México).

REPORTES DEL CENTRO CIENTIFICO IBM DE AMERICA LATINA PARA 1974.

CCAL-73-1 Levy, A.V., and Cárdenas, A.F.
"An Air Pollution Model of Mexico City"
October, 1973. To be published also at the 1974 Summer Simulation Cónference Proceedings.

CCAL-74-3 Fernández, J., Mendoza, E., and Oscós, A.
"Teaching mathematics through APL"
February, 1974.

CCAL-74-3 Lamb, R.G.
"The calculations of long period mean pollutant concentrations in an urban atmosphere".
April, 1974. To be published also in Journal of the Air Pollution Control Association.

CCAL-74-5 Levy, A.V., and Cárdenas, A.F.
"Un modelo de contaminación ambiental de la Ciudad de México".
Mayo, 1974.

CCAL-74-7 Bribiesca, E., and Avilés, R.
"Codificación en cadenas y técnicas de reducción de información para mapas y dibujos lineales"
Agosto, 1974.

CCAL-74-9 Guzmán, A., and Bribiesca, E.,
"Multiple-level geographic data bank for direct access storage".
To be submitted to the 1974 AMICEE Conference.

CCAL-74-10 Burkle, J.
"Heat-island effects in air pollution"
August, 1974.

CCAL-74-11 Howell, H.
"Substitution of labor vs. farm machines in irrigation districts in Mexico; a simulation". This is an abstract from his Ph.D. thesis in economics at the University of Pennsylvania. August, 1974. To be published also as a chapter of Analytical Studies of Mexican Agriculture, Leopoldo Solís, ed.

CCAL-74-12 Bassoco, L.M., and Howell, H.
"Tax and subsidies in Mexican farms"
September, 1974. Also to be published in referenced book.

CCAL-74-13 Howell, H., Calvo, A., and Cánovas, R.,
"On data handling for linear program modelling"
October, 1974. Also to be published in referenced book.

CCAL-74-14 Peñafiel, H.,
"User's manual of APL for S/3"
October, 1974.

CCAL-74-15 Guzmán, A., Segovia, R., and Magidin, M.
"A parallel reconfigurable LISP machine"
December, 1974.

CCAL-74-16 Lezama, H.,
"Comparisson on APL vs. Traditional notation in teaching mathematics in High Schools"
December, 1974.

CCAL-74 Bribiesca, E., and Guzmán, A.,
"User's manual for the geographic data base accessing programs".

centro de educación continua
división de estudios de posgrado
facultad de ingeniería unam

CASOS PRACTICOS EN EL DISEÑO DE SISTEMAS DE INFORMACION

BANCO DE INFORMACION SOBRE EL NIÑO MEXICANO

MANUAL DEL PROGRAMADOR DE SISTEMAS

ING. RAFAEL DOMINGUEZ

AGOSTO, 1980

SECCION 0.- GENERALIDADES DEL BANCO DE DATOS SOBRE INFORMACION DEL NIÑO MEXICANO.

0

0.1. QUE ES EL BANCO DE DATOS.

Es un conjunto de archivos que contienen información sobre el Niño Mexicano en sus tres propiedades fundamentales: Somáticas, Psíquicas y Sociales, las cuales son accesibles por medio de un sistema de programas, cuyas relaciones lógicas permiten al usuario consultar a través de pantallas, de acuerdo a las reglas indicadas en el "MANUAL DE USUARIOS DEL BANCO". Esta información son valores medidos para niños de cero a catorce años, en cada una de las tres propiedades anteriormente descritas, así como también, permite a los técnicos del Banco de Datos, el acceso para su actualización, adición, etc., requeridas para su mantenimiento de acuerdo a las reglas indicadas en este manual.

-Este conjunto de Archivos y Programas, puede trabajar en cualquier computadora; sin embargo, el diseño está orientado hacia máquinas IBM de la serie 370/, y, específicamente, se ha desarrollado esta primera versión, en la computadora IBM 370/135 del DIF.

0.1.1. OBJETIVOS DEL BANCO DE DATOS.

dices (j) y (K) "PERFIL DEL NIÑO MEXICANO" Y "ESTRUCTURA DE LOS INDICADORES DEL PERFIL DEL NIÑO MEXICANO", respectivamente. A fin de que el Banco de Datos cumpla una función más allá de sus límites para los estudiosos e investigadores sobre el Niño, y en general, para cualquier persona que desee profundizar más, sobre aspectos de interés particular al respecto, el Banco contiene información bibliográfica sobre libros, revistas, conferencias, etc., de publicaciones nacionales e internacionales, a la cual el usuario puede tener acceso rápido y sencillo, siguiendo el procedimiento indicado en el "Manual del Usuario".

Para complementar la información que contiene el Banco de Datos, se cuenta además con datos sobre encuestas, proporcionadas por las Instituciones antes mencionadas cuyo detalle, se concentra en el apéndice (H) "FUENTES DE INFORMACION".

- ORGANIZACION DE LA INFORMACION.

La organización de la información que contiene el Banco de Datos, en términos generales, se encuentra dividida en tres partes, "PARTE PARA LOS DATOS POR ENTIDADES". "PARTE PARA LOS DATOS BIBLIOGRAFICOS",

El objetivo del Banco de Datos, es brindar servicio de consulta, sobre el Niño Mexicano e información bibliográfica nacional e internacional al respecto para todas aquellas personas, que por razones de trabajo, estudios, investigación, etc., necesiten tener acceso a esta información de una manera rápida y sencilla, con todas las facilidades que les proporciona este Banco de Información, ya sea en su forma interactiva (diálogo directo con el usuario a través de una pantalla), o proceso en lote (BATCH*). Para usuarios con conocimientos de computación, brinda facilidades para elaborar pequeños programas, para la manipulación de la información, de acuerdo a objetivos específicos requeridos. Otros de los objetivos del Banco de Datos es proporcionar medios sencillos para su actualización, adición, etc., de información, por parte del personal encargado del mantenimiento del Banco.

0.1.2. INFORMACION QUE CONTIENE.

Contiene información que ha sido obtenida de varias fuentes, tales como, el ISSSTE, IMSS, DIF, SSA, SPP, etc., (ver apéndice H, "FUENTES DE INFORMACION"), que se refieren básicamente a las características Somáticas, Psíquicas y Sociales del Niño Mexicano, las cuales se indican con mayor detalle en los apén-

*BATCH: Proceso en el cual el conjunto de datos a procesar entran como una unidad, en una cola de espera para su proceso.

"PARTE PARA LOS DATOS ESPECIFICOS" tal como lo indica la figura "ORGANIZACION DEL ARCHIVO ANUAL", a su vez, cada una de estas partes está también organizada en la forma como lo indican las figuras: FIGURAS "ORGANIZACION DE LOS DATOS POR ENTIDADES", los cuales se encuentran distribuidos en tres partes:

-Parte de la República Mexicana (datos globales.

-Parte para los Estados.

-Parte para los Municipios.

Esta organización permite flexibilidad en las consultas al Banco así como confiabilidad y rapidez en las respuestas, además un desarrollo de programación más sencillo, tanto en la fase de construcción, como en el subsecuente mantenimiento, lo cual permite tener el Banco actualizado, ya sea en su información, o en cuanto a desarrollo de requerimientos futuros. FIGURA "ORGANIZACION DE LOS DATOS BIBLIOGRAFICOS", esta organización tiene todas las ventajas anteriormente descritas, además de que el usuario tiene la flexibilidad de que en la búsqueda de información puede combinar cualquiera de los campos descritos en el registro Libro i. FIGURA "ORGANIZACION DE LOS DATOS ESPECIFICOS", esta organización se refiere a las encuestas, cuya estructura se presenta en la figura, "COMO SE GUARDAN LOS ITEM DE DATOS ESPECIFICOS".

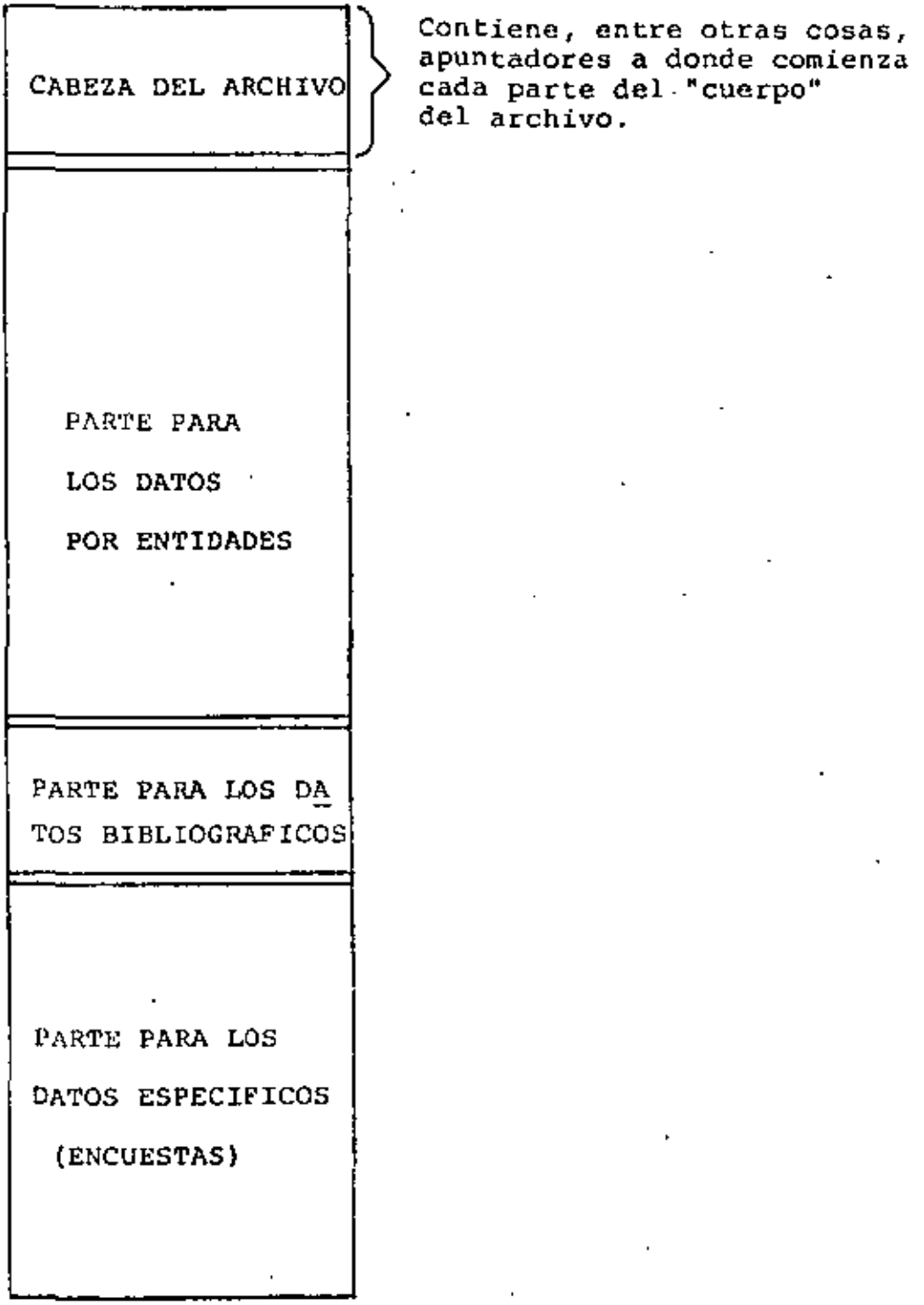


FIGURA "ORGANIZACION DEL ARCHIVO ANUAL"

Los datos de cada año se agrupan en un mismo archivo, cuya estructura está aquí descrita. Cada año habrá un archivo como el aquí indicado.

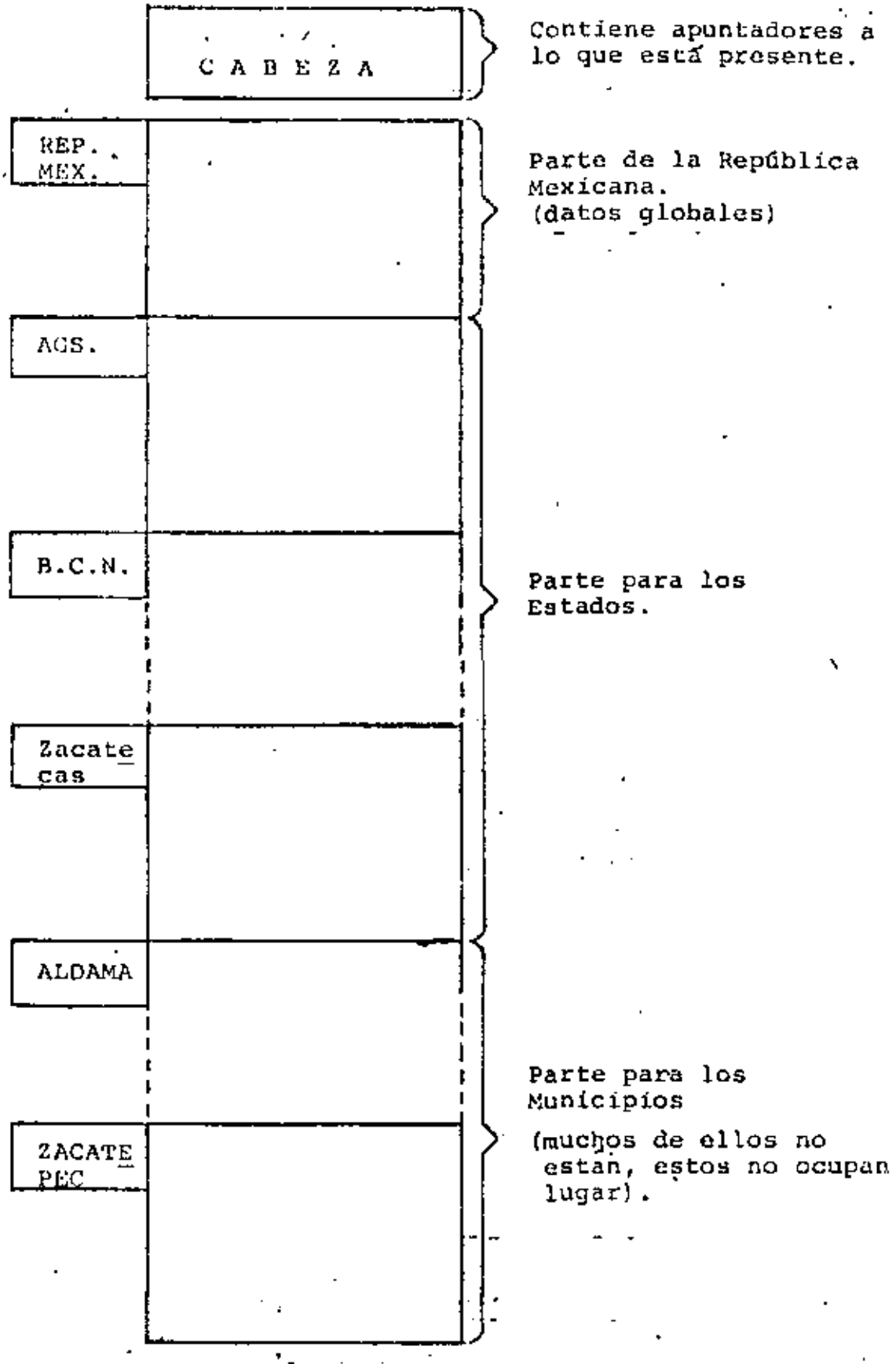
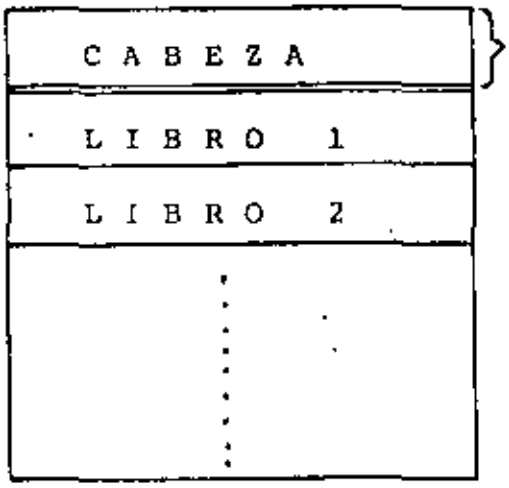


FIGURA "ORGANIZACION DE LOS DATOS POR ENTIDADES"

Son los datos nacionales, estatales y municipales, de los que se espera una repetición sistemática.



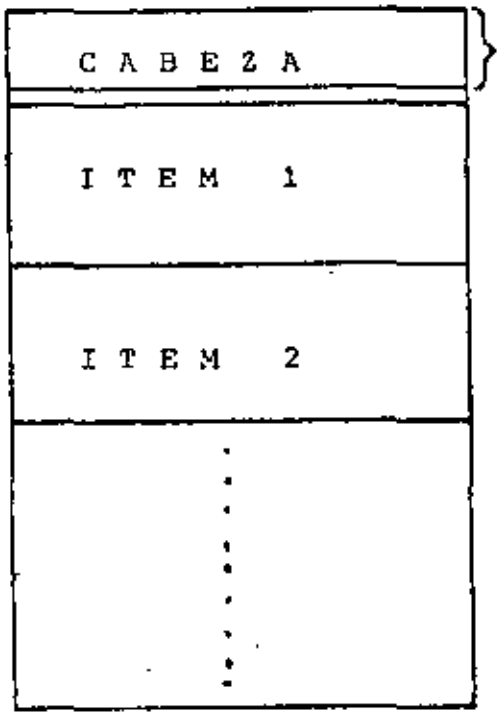
Contiene apuntadores a lo que está presente.

Libro i

# ítem	Autor	Título	Año	Editorial	# Pági-nas	Idioma	Temas o Resumen

FIGURA "ORGANIZACION DE LOS DATOS BIBLIOGRAFICOS"

Contiene la información de texto.



Contiene un directorio de donde empieza cada encuesta.

FIGURA "ORGANIZACION DE LOS DATOS ESPECIFICOS"

La parte del archivo que contiene datos específicos, su organización es como lo indica la figura.

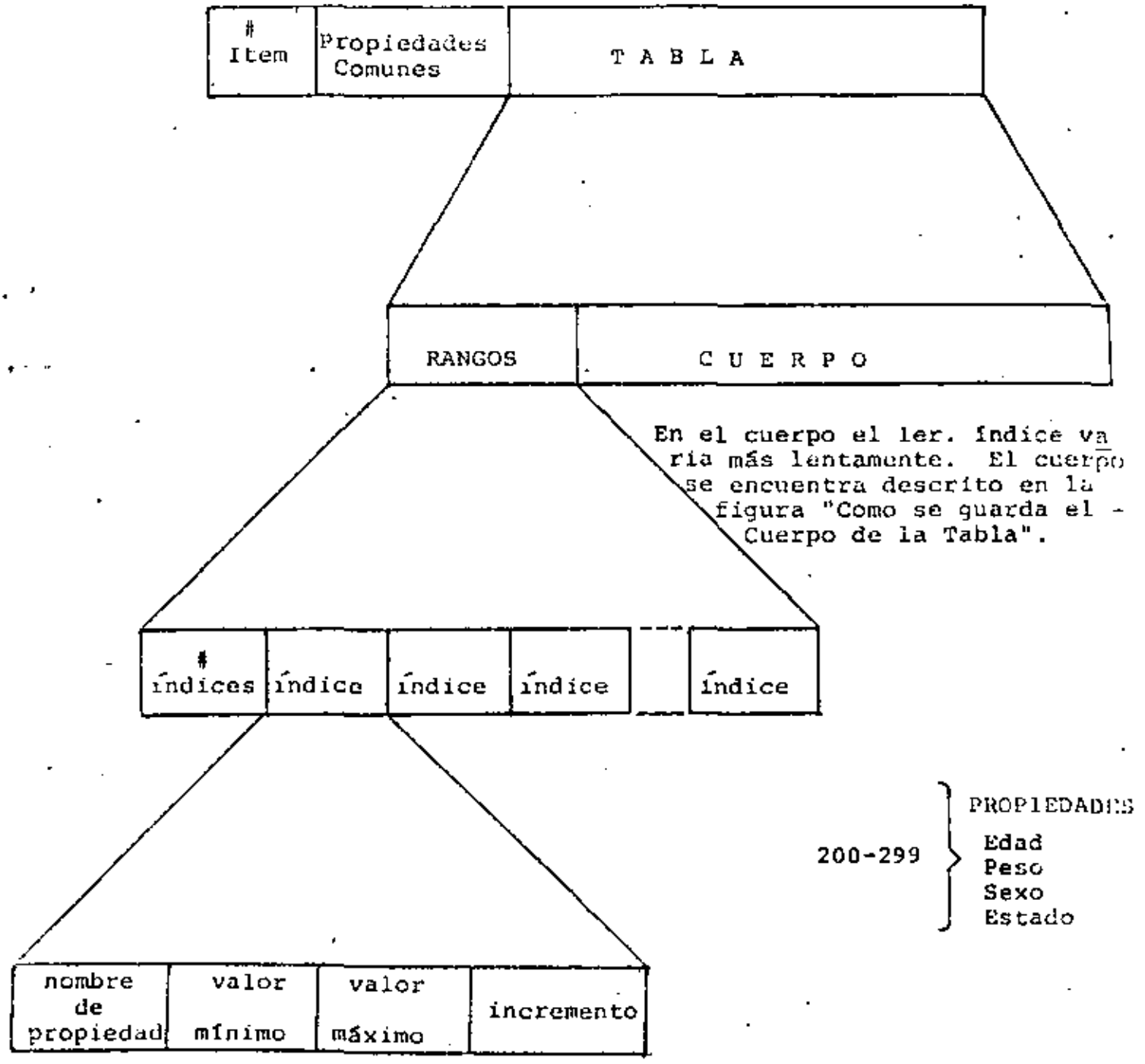


FIGURA "COMO SE GUARDAN LOS ITEMS DE DATOS ESPECIFICOS".

Varios items como el que aquí se describe, forman el cuerpo de la parte para datos específicos. En general, cada ítem corresponde a una encuesta.

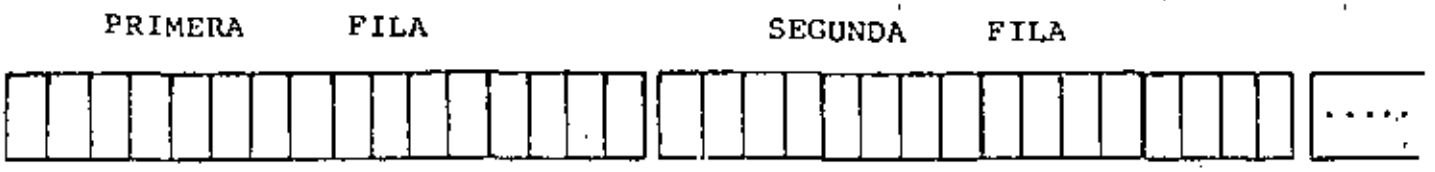


FIGURA "COMO SE GUARDA EL CUERPO DE LA TABLA"

Podemos pensar que descomponemos la tabla en tiras (filas) horizontales, que se colocan, una tras otra, en el orden vertical que tenían en la tabla.

- TIPOS DE VALORES DE LAS PROPIEDADES.

Los valores de las propiedades que se manejan en este Banco de Información son básicamente de dos tipos:

PROPIEDADES REALES.- Son todas aquellas propiedades cuyos valores almacenados, son los correctos, es decir, no es necesario someterlos a un proceso de transformación para obtener su valor verdadero. Cualquier usuario puede tener acceso a los valores de estas propiedades, con sólo seguir las reglas indicadas en el "Manual del Usuario".

PROPIEDADES COMPUTADAS.- Son aquellas propiedades que requieren de un proceso de transformación para obtener el valor correcto requerido. Estas propiedades computadas, a su vez, se dividen en dos grupos: propiedades computadas por medio de proceso criptográfico*, cuyo objetivo principal, es asegurar la información de carácter privado, en el sentido de que solamente su propietario, tiene la fórmula precisa para obtener el valor real; cualquier otro usuario, sin esta

*Ver inciso (0.2) y apéndice ("G")

fórmula obtendría un valor incorrecto. Propiedades computadas mediante fórmula matemática, el valor de estas propiedades se obtiene mediante un proceso de fórmula matemática; como por ejemplo, para obtener valores de propiedades en el sistema de medición inglés, etc.,**. Este tipo de propiedades no necesariamente es para mantener información privada, sino que puede ser utilizada para cualquier usuario, siguiendo las reglas del "Manual del Usuario".

** Su detalle se encuentra en el "Manual del Usuario".

- ALMACENAMIENTO DE PROPIEDADES.

El almacenamiento de las propiedades, para cada una de las tres partes de los archivos, se lleva a cabo en la siguiente manera :

1. Datos por Entidades : en trios de la forma

número de propiedad - valor para los hombres - valor para las mujeres.

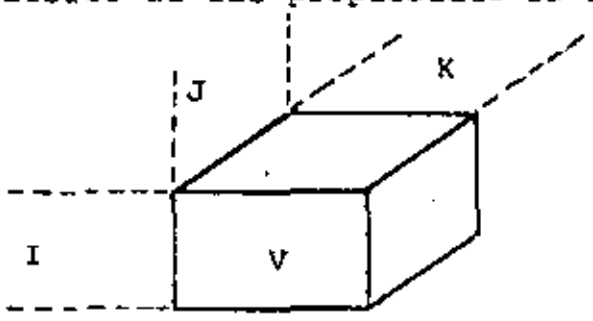
numeración de las Entidades : según apéndice "A"

Tamaño: 2 bytes - número de propiedad
2 bytes - valor de propiedad hombres
2 bytes - valor de propiedad mujeres

2. Datos Bibliográficos : por campos fijos

(propiedades implícitas)
propiedades textuales.

3. Datos específicos : El almacenamiento de los valores de las propiedades es en esta forma:



V = valor de la propiedad (2 bytes).
Indices de acceso: I,J,K

Los índices se indican en "RANGOS" de la tabla (ver FIGURA "COMO SE GUARDAN LOS ITEMS DE DATOS ESPECIFICOS"), de ahí se calcula con una fórmula, la posición de V, en el cuerpo de la tabla.

Se puede decir, que estos valores están guardados por posición calculada.

- PREGUNTAS QUE RESUELVE.

El Banco de Datos, en términos generales puede responder a cualquier pregunta que se le haga, relacionada con la población infantil de México, y en general, puede tomar una forma compleja, que consiste en una combinación de formulaciones mediante las conjunciones "o" (disyuntiva) e "y" (copulativa), su significado es el indicado en las reglas gramaticales correspondientes, y su uso se lleva a cabo de acuerdo a las reglas indicadas en el "Manual del Usuario" y a la información que el Banco contiene, (inciso 0.1.2).

A continuación se describen algunas de las preguntas que se pueden formular al Banco de Datos:

- Qué información producto de encuestas se tiene
- Qué estado de la República presenta la mayor tasa, niños-televisión.
- Cuánta población menor de un año del Distrito Federal padeció en 1975 Influenza, Neumonías, Enteritis y otras enfermedades diarréicas.
- Cuántos municipios cuentan con población menor a los 10,000 habitantes.
- Cuánta población menor de quince años hay en el país, cuyos padres de familia son profesionistas.
- En qué entidad federativa hay más niños en sexto año de primaria, menos grupos escolares y mortalidad muy alta.

- Cuánta población menor de 15 años con alteraciones de la inteligencia hay en el Distrito Federal que asisten a escuelas federales.
- Cuánta población menor de 15 años hay en Aguascalientes que anda descalza y cuánta que asiste a la escuela primaria.
- 1) Cuántas personas del sexo femenino hay en Querétaro, menor de 1 año. Y 2) En qué cantidad federativa hay menos mujeres menores de 1 año..
- Cuál es la bibliografía que existe para características antropológicas.
- Cuántos niños atípicos (alteraciones) existen por Estado.
- Cuál es el número de cunas por Estado para el IMSS e ISSSTE.
- Cuál es la tasa de mortalidad por sexo, edad y por estado, para todo el país.
- Cuál es la dieta del niño por región y su talla.
- Cuál es la atipicidad (alteración) más frecuente.
- En qué estado hay menos maestros de primaria.
- Cuántos documentos hay que se refieren a vacunas aplicadas a niños.
- Qué periodo se cubre sobre vacunación infantil.
- Con qué información se cuenta del IMSS.
- Sobre el tema de educación qué variables hay.
- Qué información existe acerca de la población indígena.

- Sobre educación, cuántos niveles de escolaridad se cubren.
- Sobre nutrición qué características se miden en cada región.
- Cuántos niños indígenas reciben enseñanza primaria, listarlos por estado y por grupos linguísticos.
- Listar las principales causas de morbilidad en el país, número de casos.
- A cuántos menores prestó atención el Hospital del Niño DIF, listarlos por año 1977, 1978.
- Qué cantidad de libros de texto gratuito se han repartido a los alumnos en los últimos cinco períodos.
- Cuántas personas menores de 15 años pertenecen a la PEA, listarlos por entidad.
- Cuál es el porcentaje de población amparada por estado, en el IMSS y en el ISSSTE.
- Qué entidad cuenta con el mayor número de viviendas con radio.
- Dónde hay mayor número de niños de 4 a 5 años y cuál es el número total de alumnos en el nivel preescolar para esa Entidad Federativa.
- En qué entidad federativa existe la mayor tasa de mortalidad.

  Ahora en esa entidad cuál es la población económicamente activa de 12 a 14 años.

  Qué población de 6 a 14 años asiste a la escuela.

  Cuál es el índice de analfabetismo, etc.

- Cuántas entidades del país se encuentran en condiciones precarias (menos del 50% de condiciones de bienestar) con respecto a los siguientes rubros y dar los procentajes.
  Viviendas propias
  Viviendas cuyo material predominante en pisos, muros y techo es: etc.
  La pregunta anterior se puede hacer para todas las variables que se quieran considerar.
- Quiero saber todo lo referente a una X entidad y después dar la comparación entre esta entidad y las demás y que peso tiene esta entidad dentro de los factores que hacen que el país prospere.
- Quiero las tasas de crecimiento del país que tenga el Banco y además una estimación para la última década de los XX.
  Esta pregunta también se puede hacer para todas las variables económicas que se tengan.
- Quiero saber cuántas vacunas se aplicaron de poliomielitis, sarampión, tos ferina, etc.
  Cuántas defunciones se presentaron por estas enfermedades y la relación entre éstas.
  (Qué tan bien funcionan las campañas de vacunación).
  Para educación se puede hacer una pregunta semejante, referente a la atención que se presta a los niños.
- En qué entidades federativas se tiene un porcentaje menor al 50% con respecto a viviendas que disponen de energía eléctrica.

0.1.3.- MODOS DE USO DEL BANCO DE DATOS.

El Banco de Datos, tiene dos modos de uso: EL INTERACTIVO (diálogo directo con el usuario a través de una pantalla de video); el BATCH, (por lotes).

El modo interactivo es el modo rápido para consulta de usuarios, ya que permite el diálogo en el mismo instante que se esté accesando, teniendo además la opción (modo batch), de mandar imprimir, las respuestas que le sean de interés particular. Los usuarios con conocimientos de computación pueden elaborar programas, a fin de obtener resultados complejos específicos. (*).

El modo batch, permite acceso igual que en el modo interactivo, con la diferencia de que la respuesta es un lote, de acuerdo al orden que le corresponda su proceso. Además, este medio es el usual, para el mantenimiento del Banco, adicionar, cambiar, etc., ya sea datos o expansión del Banco, así como los cambios en los parámetros de la información privada o secreta, lo cual sólo podrán hacerlo los usuarios autorizados. (*).

- COMO SE OBTIENE ACCESO A LA INFORMACION.
  Para obtener acceso a la información, es necesario cumplir con los procedimientos indicados por el centro de cómputo donde reside el Banco de Datos, además de auxiliarse del Manual del Usuario, así como del presente manual.

(*) Consultar el Manual de Usuario, donde se detallan ambos métodos de uso.

(*) Ver inciso 0.2.

0.2.- SEGURIDAD DE LA INFORMACION PRIVADA ("DATOS VALIOSOS") EN EL BANCO DE INFORMACION SOBRE EL NIÑO MEXICANO.

En virtud de que el Banco de Información sobre el Niño Mexicano, es abierto para cualquier tipo de usuario, es necesario proteger la información privada "DATOS VALIOSOS", de manera de que únicamente puedan obtenerlos los "USUARIOS AUTORIZADOS". Esta protección se ha llevado a cabo mediante el uso de técnicas criptográficas descritas en el apéndice "G", ALGORITMOS DE CRIPTOGRAFIA".

A continuación se describe brevemente este sistema.

"DATOS VALIOSOS"

Se nombra "DATOS VALIOSOS" a la cantidad de datos que han sido recopilados con mucho tesón y perseverancia, con patrocinio del gobierno mexicano o privado, durante varios años o lustros por investigadores nacionales, por lo que son actualmente indispensables para una explotación eficaz de la información sobre el niño mexicano.
Para que estos datos puedan ser depositados de una manera confiable en el Banco de Información, se requiere una gran seguridad, en cuanto a que no vayan a ser usados por ninguna persona si no es con la autorización expresa y así evitar fugas y manejo no idóneo de los datos.

"DUEÑOS DE LOS DATOS"

En este apartado se denomina "DUEÑO DE LOS DATOS" a aquellas personas que ejercen tutelaje sobre el uso de cierta información, debido a que ellos las han estado recopilando como parte de sus investigaciones, o que consideran que su explotación debe estar bajo su autorización personal.

Los "dueños de los datos" serán los únicos que podrán autorizar el uso de esta información (que se encontrarán grabadas en el Banco de Información Sobre el Niño Mexicano) a quienes ellos deseen. El Banco de Información deben poseer protecciones eficientes para evitar que personas no autorizadas puedan usar los "datos valiosos".

"USUARIOS AUTORIZADOS"

Como "usuarios autorizados" se consideran a los usuarios que han logrado persuadir a los "dueños de los datos" de que les permitan usar los "datos valiosos".

Para un usuario autorizado, el uso de estos datos no debe presentar problema alguno, pudiéndose desde luego mezclarse con otros datos "comunes" de acceso al público en general.

METODO DE ALMACENAMIENTO

Cada "dato valioso" se almacenará transformado en otro, desfigurado o metamorfoseado, de tal suerte que esta transformación lo convierta en otro dato totalmente ininteligible e inútil.

Ejemplo de datos definidos previamente como valiosos.

longitud de los dedos	6.2 cm.
longitud de las uñas	1.1 cm.
longitud del pelo	35. cm.
perímetro del vientre	80. cm.

Serán almacenados de la siguiente manera :

longitud de los dedos	98
longitud de las uñas	58
longitud del pelo	330
perímetro del vientre	690

Y en esta forma "desfigurada" estarán accesibles a cualquier persona. Se entiende que en esta forma desfigurada no les sirven a nadie. Para que cobren sentido se necesita saber cuál fué el procedimiento de transfiguración ó metamorfosis experimentada por éllos.

COMO SE AUTORIZA SU USO

Cuando el "dueño de los datos" es convencido de la legitimidad, conveniencia, etc., de uso de estos datos por un -- "usuario autorizado", lo autoriza efectivamente a usarlos, - diciéndole el procedimiento de transformación que los vuelve inteligibles.

En el ejemplo anterior, para pasar de un "dato valioso" a un "dato desfigurado" se multiplicó cada dato valioso por ocho, y luego se le agregó 50; obteniéndose.

longitud de los dedos	6 cm x 8 + 50 = 48 + 50	= 98
longitud de las uñas	1 cm x 8 + 50 = 8 + 50	= 58
longitud del pelo	35 cm x 8 + 50 = 280 + 50	= 330
perímetro del vientre	80 cm x 8 + 50 = 640 + 50	= 690

Entonces, el procedimiento de transformación para recuperar los "datos valiosos" es la clave o contraseña "RESTALE - CINCUENTA Y LUEGO DIVIDELO ENTRE OCHO".

Por ejemplo, del dato inservible

"perímetro del vientre" = 690

se obtiene: RESTALE CINCUENTA: 690 - 50 = 640

DIVIDELO ENTRE OCHO: 640 ÷ 8 = 80

y así se recupera el "dato valioso" original 80.

La clave "RESTALE CINCUENTA Y LUEGO DIVIDELO ENTRE OCHO"

- no va a estar almacenada en el Sistema de Información.
- no le van a conocer los programadores, analistas, etc.
- se puede cambiar por el "dueño de los datos" tan a me

pido como se estime necesario.

- sólo la va a conocer el "dueño de los datos" y los "usuarios autorizados" (autorizados por el "dueño de los datos").

Los "datos valiosos" no van a estar guardados en el -- Banco. Los "datos transfigurados" son los que van a estar - guardados en el Banco, pero sólo los poseedores de la "clave" (procedimiento de transformación) podrán hacer uso coherente y racional de esos datos.

DIFICULTAD DE LA CLAVE

En el ejemplo se ha puesto un "procedimiento de transformación" sencillo (multiplícalo por ocho y luego añádele cincuenta), pero es fácil comprender que estos procedimientos -- pueden ser arbitrariamente complejos, de manera que sea prácticamente imposible descifrarlos sin saber la clave.

COMO SE USA UN DATO VALIOSO

Cualquier usuario (autorizado o no) puede usar cualquier dato, incluyendo los "datos transfigurados".

Para des-transfigurar un "dato transfigurado", es decir, para hacerlo entendible, el "usuario autorizado" proporciona, junto con su pregunta, una tarjeta perforada (él mismo la -- perfora, para evitar que su secretaria o ayudante conozca la clave y la pueda vender) al sistema de información. Si esta clave es la correcta, los "datos transfigurados" se volverán inteligibles. Si esta clave es incorrecta, los "datos transfigurados" continuarán ininteligibles.

FRECUENCIA DE CAMBIO DE LA CLAVE

El "dueño de los datos" podrá cambiar la clave fácilmente, tantas veces como quiera y con el grado de complejidad - que deseé aplicarle.

RANGO DE LOS CODIGOS DE LAS PROPIEDADES PRIVADAS.

Los códigos de las propiedades privadas o secretas, van del número 10 001 al 10 500, como valor que se encuentra almacenado, el cual no es valor real; como valor a usar, para estas propiedades secretas, tienen los códigos que van del 15 001 al 15 500, es decir, se le suma 5 000, como primer paso, para poder iniciar el diálogo del "USUARIO AUTORIZADO" con el sistema, y, a continuación se darán los argumentos de la fórmula de transformación, para obtener el valor correcto de la información que se solicita.

2.1.- ESTRUCTURA GENERAL DEL ARCHIVO CONAIN (BANCO DE DATOS)

"CABEZA"

-1	CABEZA	# PROPIEDAD	D I R E C T O R I O

"ENTIDADES FEDERATIVAS"

-2	ENTIDAD FEDERATIVA C A B E Z A	D I R E C T O R I O	
-5	CABEZA	CUERPO REGISTRO	
	-5	CABEZA	CUERPO REGISTRO
-6	CABEZA	CUERPO REGISTRO	
-2	CABEZA	DIRECTORIO	
-5	CABEZA	CUERPO REGISTRO	
	-5	CABEZA	CUERPO REGISTRO
-6	CABEZA	CUERPO REGISTRO	

"ENCUESTAS"

-3	ENCUESTAS CABEZA	DIRECTORIO	
-7	CABEZA	CUERPO	
	-7	CABEZA	CUERPO
-3	CABEZA	DIRECTORIO	
-7	CABEZA	CUERPO	

"T E X T O"

-4	TEXTO CABEZA	CUERPO REGISTRO	-4	CABEZA	CUERPO
-4			-4		
-4					

REGISTRO FISICO → 1000 BYTES

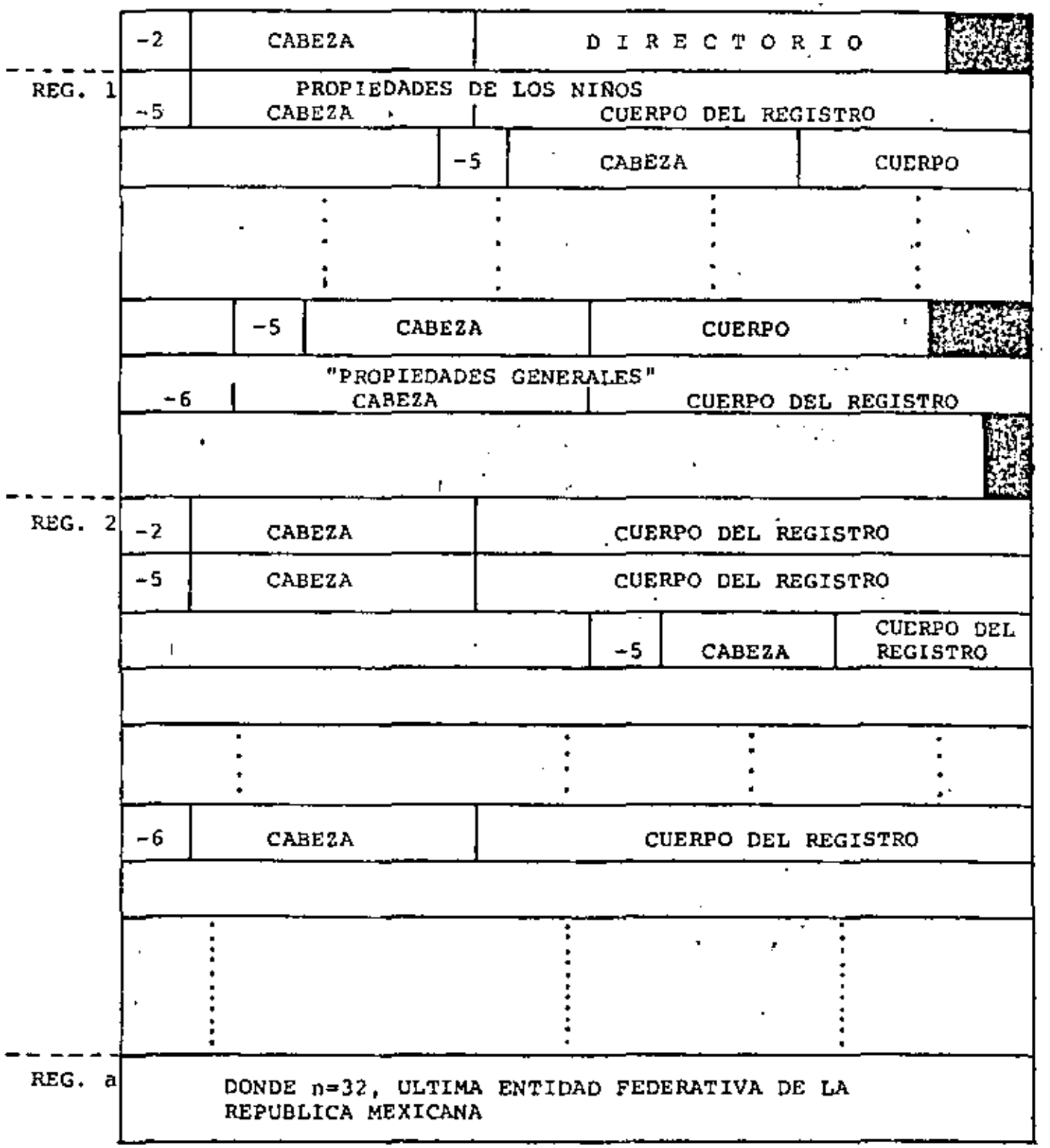
2.1.1.- ESTRUCTURA PARA LA PARTE DE ENTIDADES FEDERATIVAS

CADA ENTIDAD FEDERATIVA tiene tres tipos de registros:

-2 = Directorio

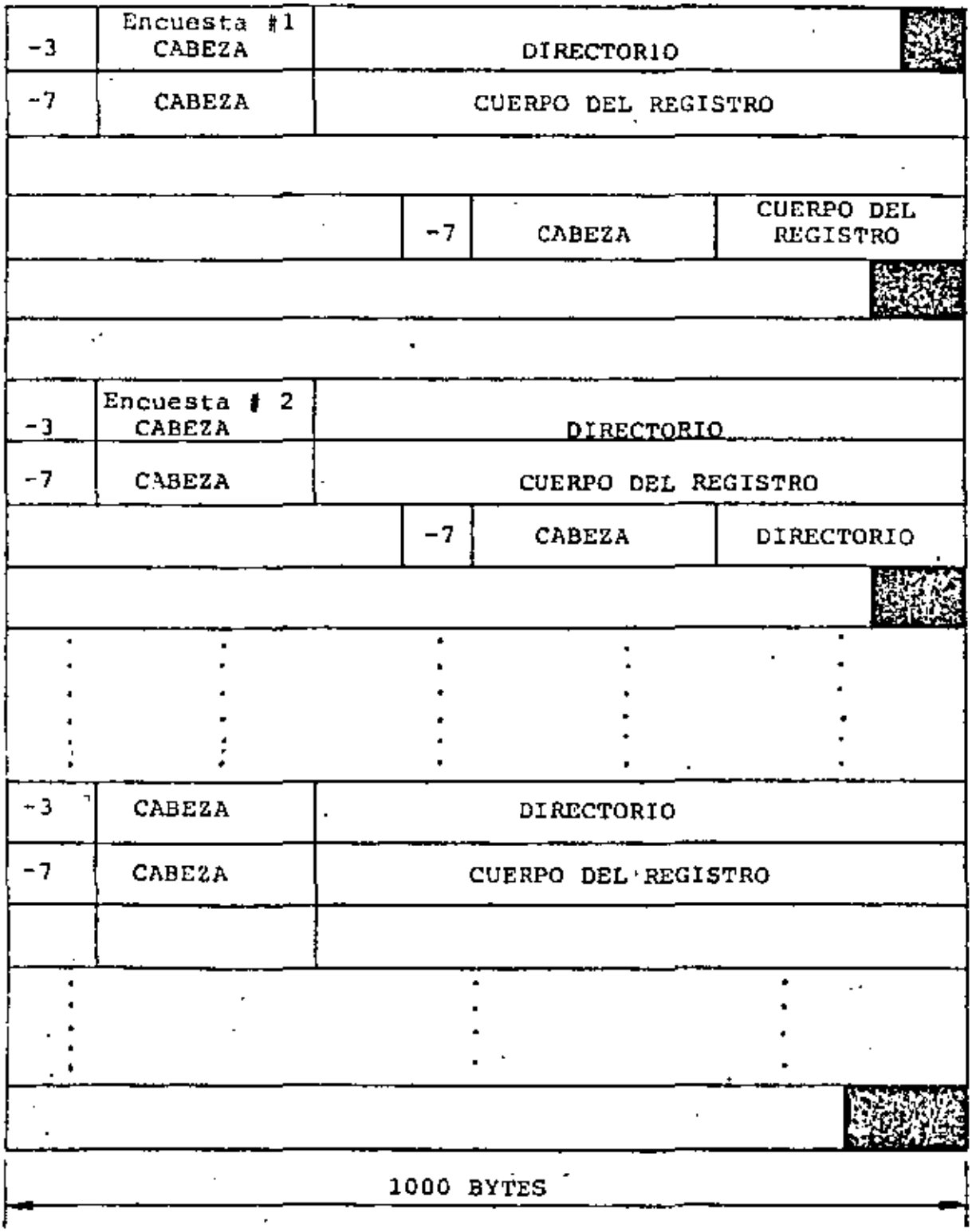
-5 = Cuerpo del registro (propiedades de los niños)

-6 = Propiedades generales.



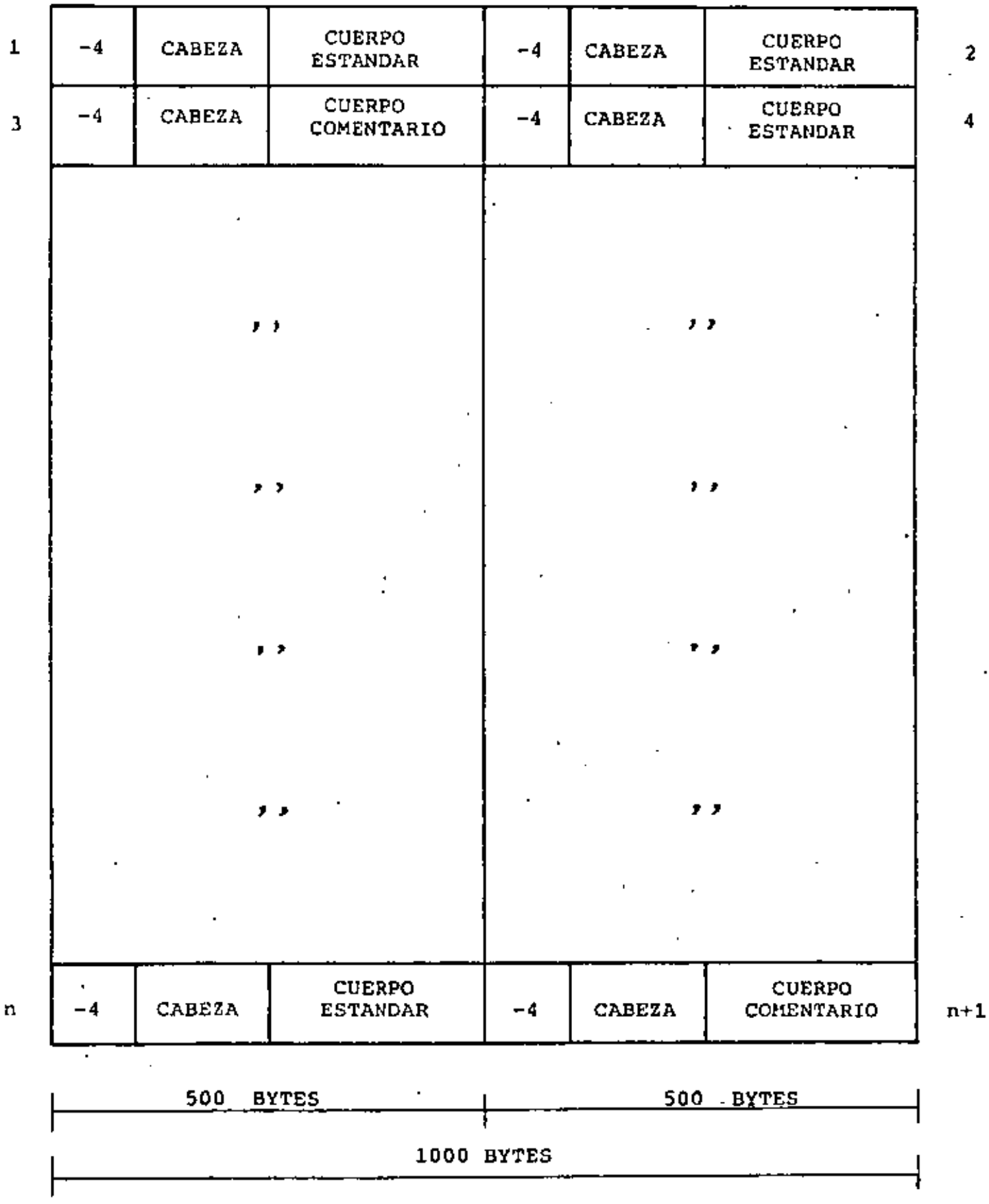
espacio no disponible o no utilizable

2.1.3.- ESTRUCTURA PARA LA PARTE DE ENCUESTAS



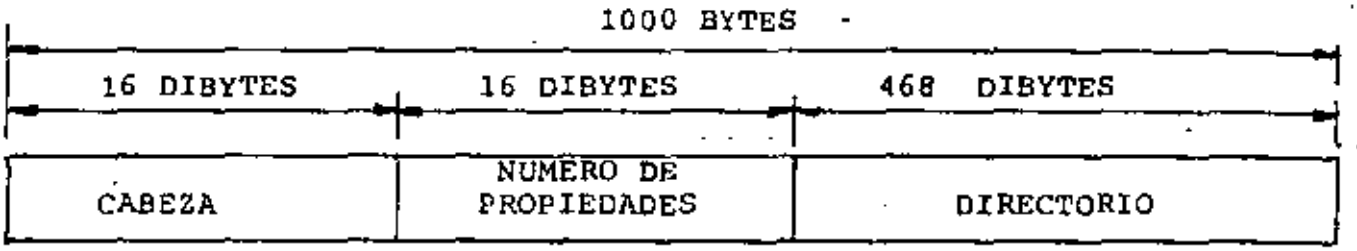
Espacio disponible o no utilizable.

2.1.3.- ESTRUCTURA PARA LA PARTE DE DATOS BIBLIOGRAFICOS (TEXTO)



2 REGISTROS LOGICOS POR REGISTRO FISICO

2.1.4 DESCRIPCION DEL REGISTRO MAESTRO.



"C A B E Z A"

NUM DE CAMPO	D E S C R I P C I O N	CANTIDAD (EN BYTES)
1	INDICADOR DEL TIPO DE REGISTRO (1)	2
2	LONGITUD TOTAL DEL ARCHIVO	2
3	NUMERO DE REGISTROS FISICOS DEL REGISTRO MAESTRO	2
4	NUMERO DE ENTRADAS EN EL DIRECTORIO	2
5	LONGITUD TOTAL DE ENTIDADES FEDERATIVAS	2
6	NUMERO DE ENTIDADES FEDERATIVAS	2
8	POSICION DE LA PRIMERA ENTIDAD FEDERATIVA	2
9	NUMERO DE ENCUESTAS	
10	POSICION DE LA PRIMERA ENCUESTA	2
11	LONGITUD TOTAL DE TEXTOS	2
12	NUMERO DE TEXTOS	2
13	POSICION DEL PRIMER TEXTO	2
14	AÑO	2
15		2
16	CANTIDAD DE TIPO DE NUMERO DE PROPIEDADES	2

"N U M E R O D E P R O P I E D A D E S"

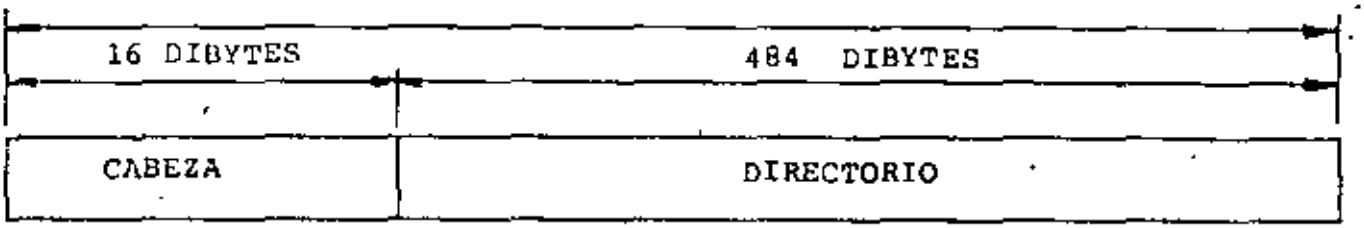
NUM. DE CAMPO	D E S C R I P C I O N	CANTIDAD (EN BYTES)
17	-	2
18	-	2
19	-	2
20	-	2
21	-	2
22	-	2
23	-	2
24	-	2
25	-	2
26	-	2
27	-	2
28	-	2
29	-	2
30	-	2
31	-	2
32	-	2

"DIRECTORIO"

1	POSICION DEL REGISTRO LOGICO
2	LONGITUD DEL REGISTRO LOGICO
	(REGISTRO 1 = 468 ENTRADAS)
	(REGISTRO 2, 3 .....n = 500 ENTRADAS)

2.1.5 DESCRIPCION DEL REGISTRO PARA LA PARTE DE ENTIDADES FEDERATIVAS.

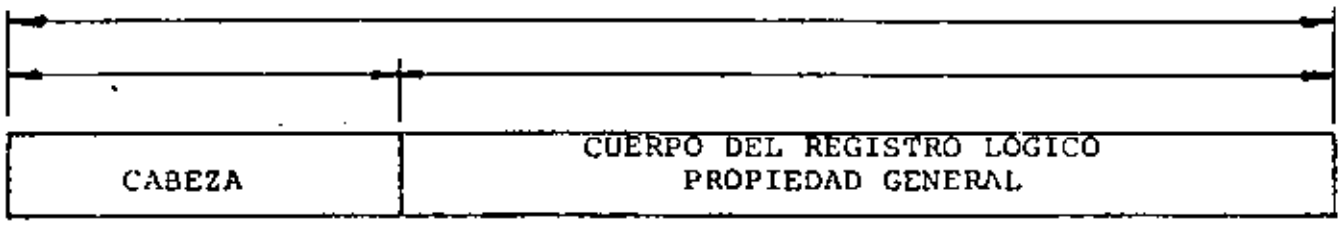
- DIRECTORIO (-2)



" CABEZA "

NUM DE CAMPO	D E S C R I P C I O N	CANTIDAD (EN BYTES)
1	INDICADOR DEL TIPO DE REGISTRO (-2)	2
2	NUMERO DE ENTIDAD FEDERATIVA	2
3	NUMERO DE REGISTROS FISICOS	2
4	NUMERO DE REGISTROS LOGICOS DE EDADES	2
5	POSICION INICIAL DEL REGISTRO DE EDADES	2
6	POSICION FINAL DEL REGISTRO DE EDADES OCUPADAS	2
7	POSICION FINAL DE PROPIEDADES DE EDADES OCUPADAS (APUNTA AL FINAL DE UN REGISTRO FISICO)	2
8	POSICION INICIAL DE PROPIEDADES GENERALES	2

- PROPIEDADES GENERALES



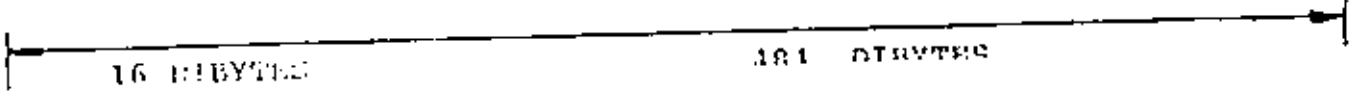
"C A B E Z A".

NUM. DE CAMPO	D E S C R I P C I O N	CANTIDAD (EN BYTES)
1	INDICADOR DEL REGISTRO LOGICO GENERAL -6	2
2	NUMERO DE ENTIDAD FEDERATIVA	2
3	CODIGO INDICADOR DE PROPIEDAD GENERAL	2
4	LONGITUD TOTAL EN DIBYTES DE ENTIDADES REGISTRO LOGICO	2
5	POSICION FINAL PARA PROPIEDADES IMPLICITAS (POSICION MAXIMA)	2
6	NUMERO DE PROPIEDAD IMPLICITA INICIAL	2
7	NUMERO DE PROPIEDAD IMPLICITA FINAL	2
8	POSICION FINAL PARA PROPIEDADES EXPLICITAS ORDENADAS (POSICION MAXIMA)	2

32

2.1.5 DESCRIPCION DEL REGISTRO PARA LA PARTE DE ENTIDADES FEDERATIVAS.

- DIRECTORIO (-2)



NUM. DE CAMPO	D E S C R I P C I O N	CANTIDAD (EN BYTES)
9	NUMERO DE PROPIEDAD EXPLICITA FINAL	2
10	POSICION DE LA ULTIMA PROPIEDAD EXPLICITA ORDENADA	2
11	POSICION DE LA ULTIMA PROPIEDAD EXPLICITA DESORDENADA	2

"CUERPO DEL REGISTRO LOGICO"
(PROPIEDAD GENERAL)

PROPIEDADES IMPLICITAS VALORES

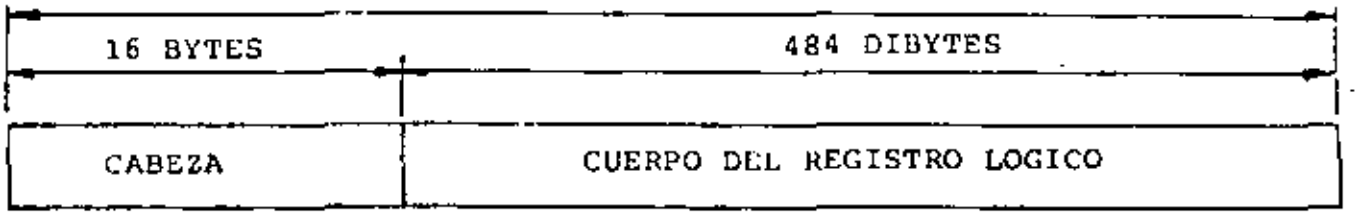
PROPIEDADES EXPLICITAS NUMERO DE PROPIEDADES VALORES.

PAQUETES DE 2X2 PARA PROPIEDADES IMPLICITAS.

PAQUETES DE 2X3 PARA PROPIEDADES EXPLICITAS.

- CUERPO DEL REGISTRO (PROPIEDADES DE LOS NIÑOS) (-5)

1000 BYTES



" C A B E Z A "

NUM. DE CAMPO	D E S C R I P C I O N	CANTIDAD (EN BYTES)
1	INDICADOR DE REGISTRO LOGICO DE EDAD	2
2	NUMERO DE ENTIDAD FEDERATIVA	2
3	NUMERO DE EDAD	2
4	LONGITUD TOTAL EN BYTES DE ESTE REGISTRO LOGICO	2
5	POSICION FINAL PARA PROPIEDADES IMPLICITAS (POSICION MAXIMA)	2
6	NUMERO DE PROPIEDADES IMPLICITAS INICIAL	2
7	NUMERO DE PROPIEDADES IMPLICITAS FINAL	2
8	POSICION FINAL PARA PROPIEDADES EXPLICITAS ORDENADAS (POSICION MAXIMA)	2
9	NUMERO DE PROPIEDADES EXPLICITAS FINAL	2
10	POSICION DE LA ULTIMA PROPIEDAD EXPLICITA ORDENADA VACIA	2

11	POSICION DE LA ULTIMA PROPIEDAD EXPLICITA DESORDENADA VACIA	2
.		.
.		.
.		.
.		.
.		.
16		2

"CUERPO DEL REGISTRO"

PROPIEDADES IMPLICITAS

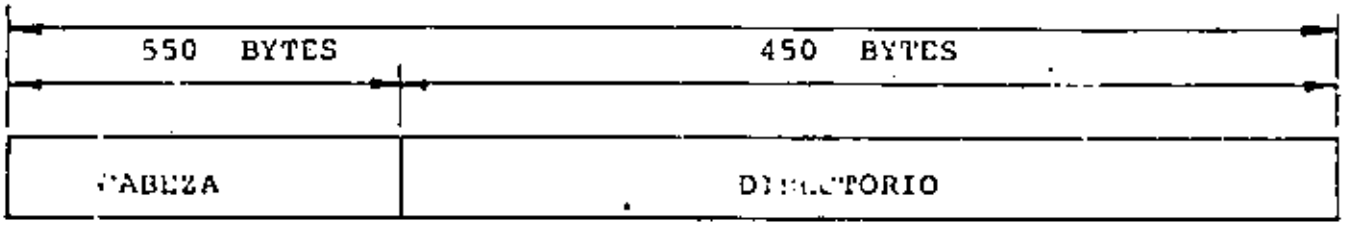
1	VALORES PARA HOMBRES	

NUM. DE CAMPO	D E S C R I P C I Ó N	CANTIDAD (EN BYTES)
2	VALORES PARA MUJERES	
	O	
3	VALORES PARA HOMBRES Y MUJERES	
4	INDICADOR DE INFORMACION AGRUPADA	

PROPIEDADES EXPLICITAS

5	NUMERO DE PROPIEDAD	
6	VALOR PARA HOMBRES	
7	VALOR PARA MUJERES	
	O	
8	NUMERO DE PROPIEDAD	
9	VALOR PARA HOMBRES Y MUJERES	
10	INDICADOR DE INFORMACION AGRUPADA	
	PAQUETES DE 2X2 BYTES PARA PROPIEDADES IMPLICITAS.	36
	PAQUETES DE 2X3 PARA PROPIEDADES EXPLICITAS.	

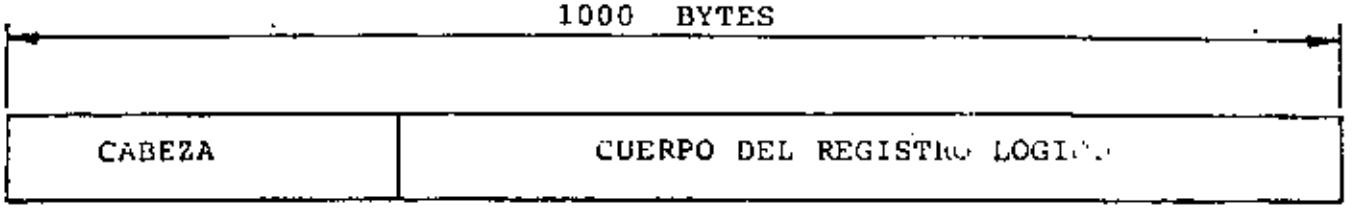
2.1.6 - DESCRIPCION DEL REGISTRO PARA LA PARTE DE ENCUESTAS.



" C A B E Z A "

NUM. DE CAMPO	D E S C R I P C I O N	CANTIDAD (EN BYTES)
1	INDICADOR DEL TIPO DE REGISTRO (= -3)	2
2	NUMERO DE ENCUESTAS	2
3	NUMERO DE REGISTROS FISICOS DE LA ENCUESTA (DE 1000 BYTES)	2
4	NUMERO DE CUADROS DE LA ENCUESTA (REGISTROS LOGICOS)	2
5	NUMERO DE ENTRADAS AL DIRECTORIO	2
6	ESPACIOS	22
7	NOMBRE DE LA ENCUESTA	80
8	INSTITUCION	80
9	AUTOR (ES)	16
10	LUGAR DONDE SE REALIZO	100
11	DURACION (EN HORAS HOMBRES O DIAS)	40
12	FECHA DE INICIO	6
13	FECHA DE PUBLICACION	6
14	ALCANCE	48

- DESCRIPCION DEL REGISTRO PARA LA PARTE DE ENCUESTAS.



NUM. DE CAMPO	D E S C R I P C I O N	CANTIDAD (EN BYTES)
1	INDICADOR DEL TIPO DE REGISTRO (= -7)	2
2	NUMERO DE CUADRO DE LA ENCUESTA	2
3	LONGITUD DEL REGISTRO LOGICO	2
4	NUMERO DE INDICES VERTICALES	2
5	NUMERO DE LA PRIMERA PROPIEDAD (MAS EXTERIOR)	2
6	NUMERO DE LA SEGUNDA PROPIEDAD (MAS EXTERIOR)	2
7	NUMERO DE LA TERCERA PROPIEDAD (MAS EXTERIOR)	2
8	NUMERO DE LA CUARTA PROPIEDAD (MAS EXTERIOR)	2
9	NUMERO DE LA QUINTA PROPIEDAD (MAS EXTERIOR)	2
10	NUMERO DE LA SEXTA PROPIEDAD (MAS EXTERIOR)	2

NUM DE CAMPO	D E S C R I P C I O N	CANTIDAD (EN BYTES)
15	ESPACIOS	58
16	DIRECTORIO (75 ENTRADAS DE 3 BYTES c/u)	450

- NUMERO DE CUADRO
- POSICION INICIAL DEL REGISTRO LOGICO
- LONGITUD DEL REGISTRO LOGICO (DIBYTES)

NUM. DE CAMPO	D E S C R I P C I O N	CANTIDAD (EN BYTES)
11	CUANTOS VALORES TIENE LA PROPIEDAD MAS EXTERIOR	2
12	2	2
13	3	2
14	4	2
15	5	2
16	6	2
.		.
.		.
.	100 VALORES DISTINTOS DE LA PROPIEDAD MAS EXTERIOR A LA SEXTA.	.
1	NUMERO DE INDICES HORIZONTALES	2
2	NUMERO DE LA PRIMERA PROPIEDAD (MAS EXTERIOR)	2
3	NUMERO DE LA SEGUNDA PROPIEDAD (MAS EXTERIOR)	2
4	NUMERO DE LA TERCERA PROPIEDAD (MAS EXTERIOR)	2
5	NUMERO DE LA CUARTA PROPIEDAD (MAS EXTERIOR)	2

2.1.7 - DESCRIPCION DEL REGISTRO PARA LA PARTE DE TEXTO.

NUM. DE	D E S C R I P C I O N	CANTIDAD (EN BYTES)
1	INDICADOR DE TIPO DE REGISTRO (TEXTO = -4)	2
2	NUMERO DE TEXTO	2
3	ESPACIO	2
4	INDICADOR DE CLASE DE TEXTO	2
5	ESPACIO	2
6	AUTOR	80
7	TITULO	80
8	CLASIFICACION	12
9	IDIOMA	20
10	EDITORIAL	60
12	PALABRAS CLAVE (RESUMEN)	232

FORMATO PARA COMENTARIO (TEXTO) 42

NUM. DE CAMPO	D E S C R I P C I O N	CANTIDAD (EN BYTES)
1	INDICADOR DE TIPO DE REGISTRO (TEXTO = -4)	2
2	NUMERO DE TEXTO	2
3	ESPACIO	2
4	INDICADOR DE CLASE DE TEXTO	2
5	LONGITUD EN BYTES DEL COMENTARIO	2
6	COMENTARIO (HASTA 490 BYTES)	490

2.2. - DIAGRAMA DE BLOQUES DE LA GENERACION PARA CARGAR LOS DATOS AL BANCO DE DATOS.

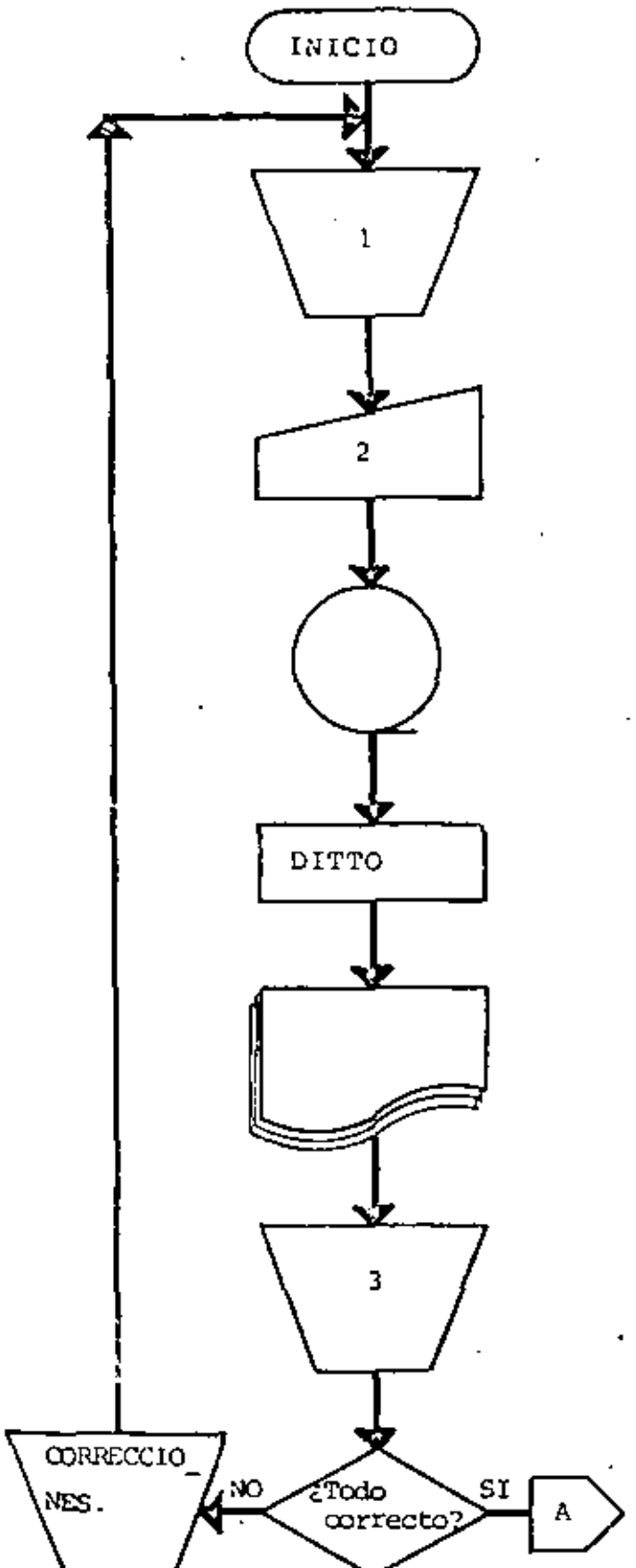
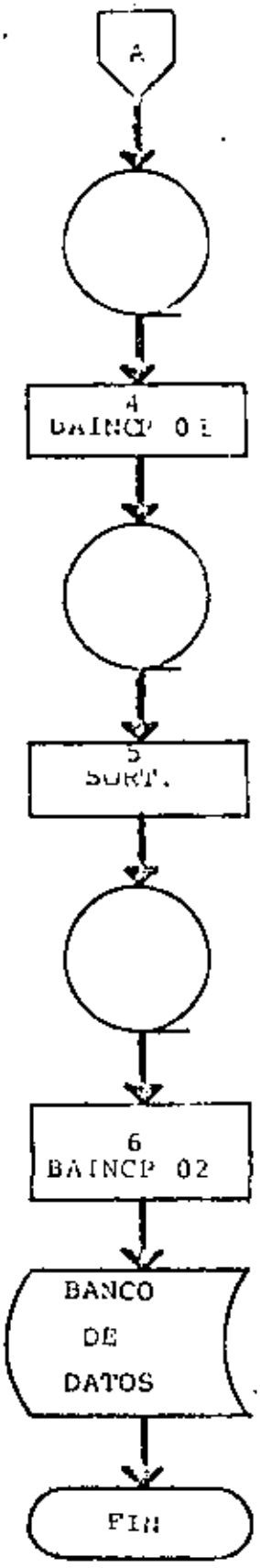
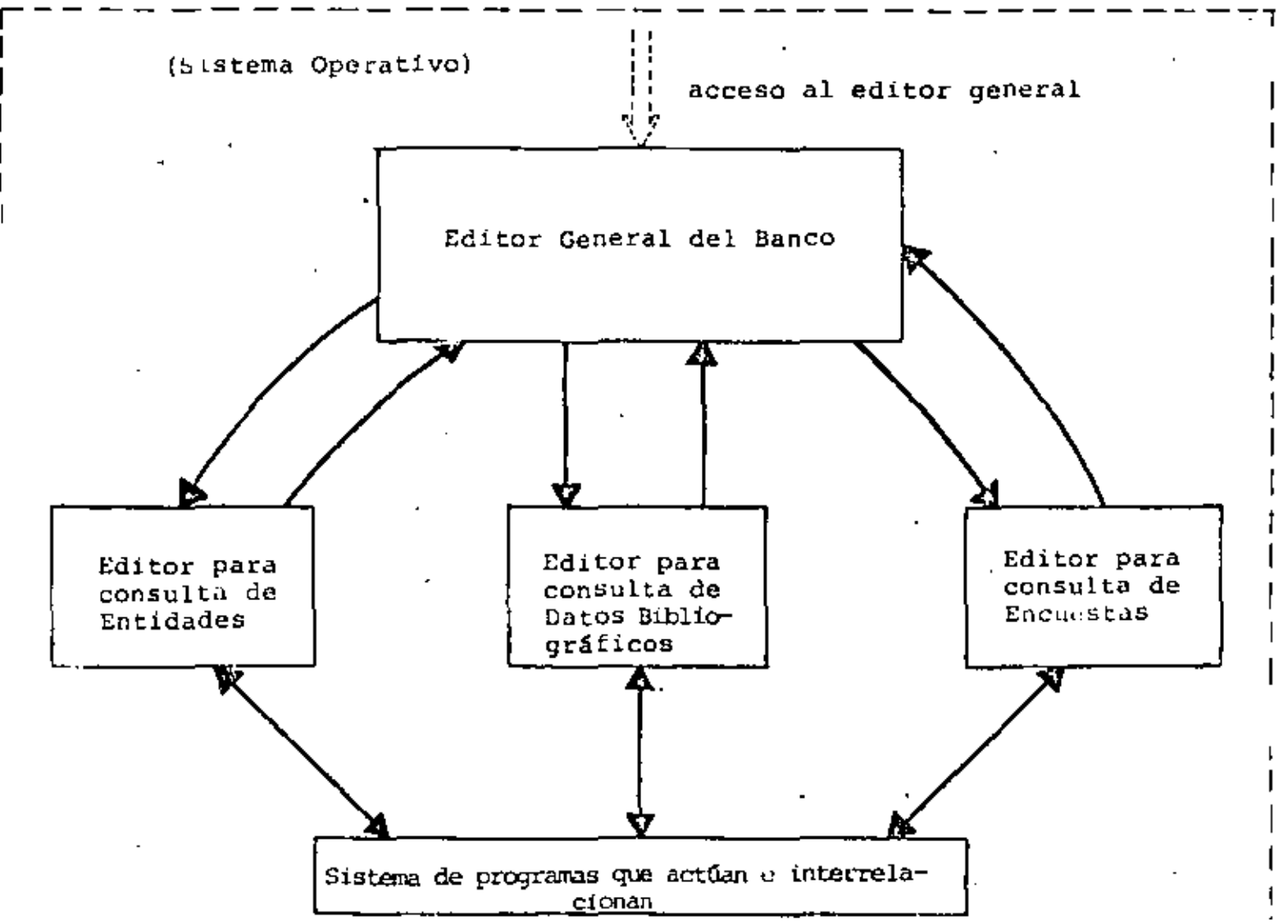
D I A G R A M A	D E S C R I P C I O N
	1.- Codificación de la información sistematizada 2.- Captura de los datos (CADE) - Cinta con la captura de datos - Listado de la cinta 3.- Validación de los datos - Datos correctos quedan en la cinta de captura - Datos incorrectos, se eliminan de la cinta de Captura de datos y se vuelve a repetir el proceso.

DIAGRAMA	DESCRIPCION
	- Cinta con registros correctos de tipo variable (5 tipos) y de 103 posiciones 4.- Programa BAINCP 01, que reformatea los registros a un solo tipo, y de 40 posiciones. - cinta con registro formateados (40 Bytes). 5.- proceso de clasificación en el siguiente orden. AÑO, ESTADO, EDAD y PROPIEDAD - cinta clasificada. 6.- programa para cargar el Banco - Datos cargados en disco

SECCION 3.- SISTEMA OPERATIVO DEL BANCO DE DATOS.

El Sistema Operativo del Banco de Datos, está integrado por una serie de programas que actuan e interrelacionan con las tres partes que integran cada uno de los archivos anuales que forman el Banco de Datos, tal como se vió y se describió en secciones anteriores, este conjunto de programas está administrado por un Editor General del Banco, el cual a su vez administra los tres partes constitutivas del archivo anual en sus correspondientes Editores, tal como se muestra a continuación:



El acceso inicial al Banco de Datos.
Se lleva a cabo por medio del Editor General del Banco, por medio del cual, se accesa a cualquiera de las tres partes que integran el archivo anual del mismo.

Desde cualquier Editor y en el momento que se requiera se puede suspender la interacción con el comando /CANCEL*

En lo subsecuente de esta sección se describen los programas que integran el Banco de Datos.

* Para mas detalle ver el Manual del Usuario.

3.2.- DESCRIPCION Y FORMATO DE LA FUNCION "ARCHI"

Esta función verifica si existe un archivo para el año dado.

ARCHI (IAÑO)

Esta función es verdadera (T) sí el archivo que corresponde al año IAÑO está presente, en caso contrario es falso (F).

Pruebas para ARCHI (iAÑO)

Objetivo: Llevar a cabo el define file de un archivo que no existe, y que no se de cuenta JCL del Sistema Operativo de equipo, sí no que se detecte hasta la hora de ejecución.

ARCHI (iAÑO) funciona como una subrutina de ARCHI (IAÑO) _

3.3.- LISTADO DE PROGRAMA DE LA FUNCION "ARCHI"

```
FOS FORTRAN IV 1ECN-FO-475 3-8        ARCHI        DATE  07/11/75    TIME  11.33.10    PAGE 0001
0001          LOGICAL FUNCTION ARCHI(IANO)
0002          INTEGER*2 IA-70
0003          DIMENSION TABANO(10)/70,71,72,73,74,75,76,77,78,79/
0004          ARCHI=.FALSE.
0005          IF(IANO.LT.TC.OR.IAÑO.GT.79)RETURN
0006          DO 1 I=1,10
0007          IF(IANO.EQ.TABANO(I))GO TO 2
0008        1 CONTINUE
0009          RETURN
0010        2 ARCHI=.TRUE.
0011          RETURN
0012          END
```

3.4.- DESCRIPCION Y FORMATO DE LA SUBRUTINA "BORRA2"

Esta subrutina llena de ceros la tabla de pregunta.

BORRA2 (ARR, I, J, V)
Llena al arreglo bidimensional ARR (I,J) de valores V.
Todos son integer X 2.

DESCRIPCION DE LA FUNCION "CHKANO"

Esta funcion verifica si el año que se le esta dando, se encuentra dentro de los limites (1970-1989)

CHKANO (AÑO)
Esta función es verdadera (T) si el año dado por el usuario es válido, y falso (F) en caso contrario.

AÑO en punto flotante es una variable, no una constante.

3.5.- LISTADO DE PROGRAMA DE LA SUBRUTINA "BORRA2"

```
CCS FORTRAN IV 3.0(M-FD-475 3-6        BORRA2          DATE  07/11/75     TIME    11.23.21      PAGE 0001
0001            SUBROUTINE BORRA2(ARR,II,JJ,KK)
0002            IMPLICIT INTEGER*2 (A-Z)
0003            INTEGER*2 ARR(1)
0004            DO 100 I=1,II
0005            DO 200 J=1,JJ
0006            ARR(I,J)=K
0007      200   CONTINUE
0008      100   CONTINUE
0009            RETURN
0010            END
```

50

3.6.- DESCRIPCION Y FORMATO DE LA FUNCION "BYTE"

51

Esta función regresa justificado a la derecha el enésimo byte de un arreglo.

BYTE (N,ARR)

Nos regresa | 8 | 8 | en un dibyte el enésimo byte del arreglo ARR, el cual es integerX2. Lo regresa (en valor de BYTE), siempre a la derecha, con ceros binarios a la izquierda, el arreglo ARR es de N bytes de largo.

3.7.- LISTADO DE PROGRAMA DE LA FUNCION "BYTE"

```
EOS FORTRAN IV 3400-FO-435 3-8          BYTE          DATE  07/13/76     TIME  11.21.07     PAGE 0001

0001                INTEGER FUNCTION BYTE*2(A,B)*4
                C   ESTA REGRESA JUSTIFICADO A LA DERECHA EL A-ESIMO BYTE DE B
0002                IMPLICIT INTEGER*2 (A-Z)
0003                INTEGER*2 MASCES/'7FFF'/,ANDBIN
0004                FLAG=0
0005                BYTE=AND(B(N/2),IB/2**
0006                IF(BYTE.GE.0)GO TO 1
0007                FLAG=1
0008                BYTE=BYTE+MASCES+1
0009          1     IF(N-N/2*2.EQ.0)GO TO 2
0010                BYTE=BYTE/256+128*FLAG
0011                RETURN
0012          2     BYTE=BYTE-BYTE/256*256
0013                RETURN
0014                END
```

3.8.- DESCRIPCION Y FORMATO DE LA FUNCION "BUSCA (Batch)"

BUSCA (AÑO, ENTIDAD, EDAD, SEXO, PRED, PROCESO)

Esta función se describe a continuación y en el siguiente orden:

AÑO: Para todos los años especificados.

ENTIDAD: Para todas las entidades especificadas.

EDAD: Para todas las edades especificadas.

SEXO: Para todos los sexos especificados.

A los items que satisfagan el predicado PRED, se les aplica el proceso PROCESO.

PRED y PROCESO son externas que armará el usuario en FORTRAN, en el cual, PRED es un predicado lógico y PROCESO en el proceso que sufriran los items que satisfagan a PRED.

Ejemplos:

AÑO	COMENTARIO
1975	Todo lo que se encuentre en el año de 1975.
0	Significa que están incluidos todos los años.
1975.77	Significa de 1975 a 1977.

ENTIDAD	COMENTARIO
13	Número de propiedad del Estado de HIDALGO.
0	Significa todos los estados de la REPUBLICA MEXICANA.
13.25	Significa de los Estados con número de propiedad 13 al 25 (Ver apéndice A).

EDAD	COMENTARIO
1	Significa edad de 0 a 1
3	Significa edad de 2 a 3
0	Significa todas las edades

EDAD	COMENTARIO
3.5	Significa las edades de (2 a 3) a (3 a 4) a (4 a 5).

SEXO	COMENTARIO
1	Sexo masculino
2	Sexo femenino
0	Todos (ambos)

BUSCA (0, 13, 0, 0, PR, PROCE)

Está indicando buscar en:

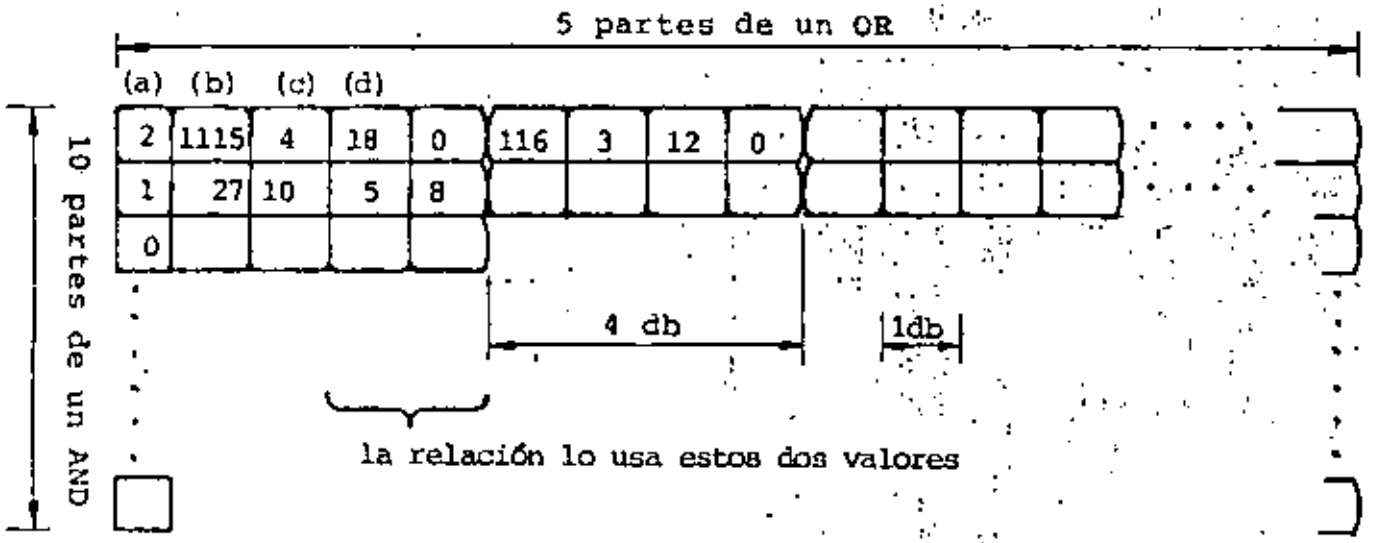
PROCE, todos los años, en el Estado de Hidalgo, en todas las edades y en todos los sexos, aquellos items que satisfagan a PR.

3.10.- DESCRIPCION Y FORMATO DE LA FUNCION "BUSCAI" (Interactivo)

BUSCAI (AÑO, ENTIDAD, EDAD, SEXO, PRED, PROCESO)

Este programa funciona como programa principal, el cual desde el teclado, lee los textos de las preguntas con el formato del arreglo PREVAL, e imprime todos los ítems (de entidades) que satisfagan la pregunta hecha, esta impresión puede llevarse a cabo ya sea por pantalla o por impresora.

Arreglo PREVAL.



(a) Indica cuantas ramas tiene el OR, o si es 0, indica que es fin.

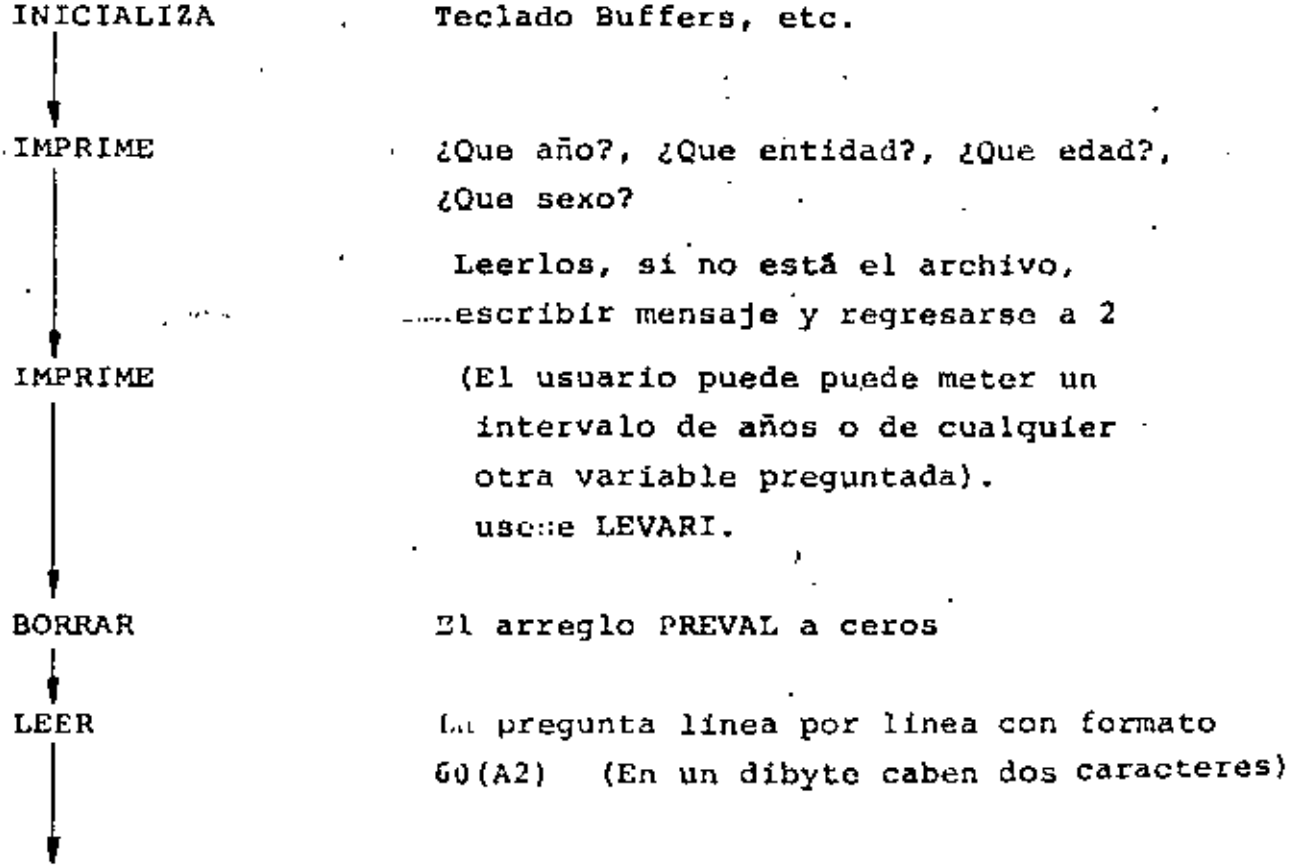
(b) Contiene el número de propiedad (del apéndice A, tabla de propiedades).

(c) Indica el número de relación, cuya tabla de conversión es la siguiente:

RELACION	NOMBRE	NUMERO
*	entre	10
<	menor que	1
=	igual a	2
>	mayor que	4
< =	menor o igual a	3
> =	mayor que o igual a	6
≠	diferente de	5
< = >	siempre es cierto	7
	siempre es falso	0

(d) Valor contra el que se compara (en binario)

-- Diagrama de FLUJO de BUSCA I (Interactivo).



LLENAR — El arreglo PREVAL línea por línea (esto se lleva a cabo con LLELIV.

PEDIR — El formato de impresión y preparar el arreglo FORIMP para su ulterior uso por el IMPRESOR.

Por cada año especificado (si está; si no enviar mensaje y continuar) el archivo correspondiente:

para cada entidad especificada.

para cada edad especificada.

para cada sexo especificado.

Ver si satisface la pregunta que está en PREVAL. Esto se hace con DIME. Imprimir con el IMPRESOR INTERACTIVO aquellos que sí lo satisfagan. Usar acá el arreglo FORIMP.
Regresarse a 2 para otra consulta.

3.13.- LISTADO DE PROGRAMA DE LA FUNCION "CHKANO"

```
COS FORTRAN IV 360N-FO-475 3-6        CHKANC          DATE  01/13/75     TIME    11.11.03      PAGE 0001
0001          LOGICAL FUNCTION CHKANO(ANO)
0002     C    SALE SI EL ANO ES VALIDO.  ANO ES REAL
0003          IMPLICIT INTEGER*2 (A-Z)
0004          REAL ANO
0005          LOGICAL LEAYR
0006          DATA IYC,IES/70,69/
0007          IF(ANO.GT.1000.0)ANO=ANO-1000.0
0008          CALL DEFPOL(ANO,51,4,20
0009          CHKANO=LEMTRE(IYC,51,IES0,ANO,LEAYR(IYC),V2,IER
0010          RETURN
0011          END
```

3.14. DESCRIPCION Y FORMATO DE LA FUNCION "CODIF"

CODIF (VAL, PAR1, PAR2, VAL2,X)

Esta función aplica al valor VAL la transformación X que usa los parámetros PAR1 y PAR2.

Propiedades "decodificadas" (computadas)	Propiedades "codificadas" (almacenadas)
P1	CODIF (g1,PAR1, PAR2,1)
P2	CODIF (g2,PAR1, PAR2,1)
.	.
.	.
.	.
.	.
.	.
Pn	CODIF (gn, PAR1, PAR2, 1)
Pn+1	CODIF (gn+1, PAR3, PAR4, 2)
.	.
.	.
.	.
.	.
.	.
Pn+k	CODIF (gn+k, PAR3, PAR4, 2)

O sea para recobrar una propiedad Pi cuando su valor codificado es gi ($1 \leq N$), se usa X=1; cuando se trata de recobrar una propiedad Pi ($n<i\leq N+K$), se usa X=2, esto se puede extender para $X>2$

3.15.- LISTADO DE PROGRAMA DE LA FUNCION "CODIF"

```
DOS FORTRAN IV 360N-FO-476 2-0          CODIF          DATE  31/13/79      TIME     17.14.51      PAGE 0001

0031                INTEGER FUNCTION CODIF*2 EVAL,PARAM1,PARAM2,TRANSF*
                    C ESTA RUTINA CODIFICA SEGUN 4 TRANSFORMACIONES DISPONIBLES
0032                IMPLICIT INTEGER*4 (A-Z)
0033                DIMENSION ITRANSF(4),ITRACE(11)
0034                AUX1=EVAL
0035                DO 5 I=1,15
0036                AUX2=AUX1/2
0037                ITRACE(I)=AUX1-AUX2*2
0038              5 AUX1=AUX2
                    C Y... DE EMPACADOS LOS 2115. SEGUN LAS DISTINTAS TRANSFORMACIONES
0039                GO TO (10,20,30,40),TRANSF
                    C ESTA TRANSFORMACION TRANSPONE LOS PARAM1 BITS DE MAS A LA DERECHA,
                    C Y MENOR QUE PARAM1 MENOR QUE 16
0040             10 IF(PARAM1.LE.1.OR.PARAM1.GE.16)GO TO 200
0041                AUX1=PARAM1/2
0042                JCPARM=1
0043                DO 15 I=1,AUX1
0044                AUX2=BITAND(ITRACE(I),JCPARM)
0045                BITAND(ITRACE(I),JCPARM)=
0046                JCPARM=JCPARM*2
0047             15 J=J-1
0048                GO TO 100
                    C ESTA TRANSFORMACION HACE UN OR EXCLUSIVO CON PARAM1
0049             20 AUX1=PARAM1
0050                DO 25 I=1,15
0051                AUX2=AUX1/2
0052                AUX1=BITAND(AUX1-AUX2*2,1)+AUX2*2
0053                BITAND(
0054                IF(AUX1.EQ.1)BITAND(
0055             25 AUX1=AUX2
0056                GO TO 100
                    C ESTA TRANSFORMACION ROTA A LA DERECHA PARAM1 VECES
                    C Y MENOR QUE PARAM1 MENOR QUE 15
0057             30 IF(PARAM1.LE.0.OR.PARAM1.GE.15)GO TO 200
0058                DO 35 I=1,15
0059             35 BITAND(
0060                J=16-PARAM1
0061                DO 40 I=1,15
0062                BITAND(
0063                IF(J.EQ.15)J=0
0064             40 J=J+1
0065                GO TO 100
                    C ESTA TRANSFORMACION TRANSPONE LOS PARAM1 BITS DE LA IZQUIERDA Y LOS
                    C PARAM2 BITS DE LA DERECHA.
                    C Y MENOR QUE PARAM1 + PARAM2 MENOR QUE 16
0066             45 AUX1=PARAM1+PARAM2
0067                IF(AUX1.LE.1.OR.AUX1.GE.16)GO TO 200
0068                IF(PARAM1.EQ.1.AND.PARAM2.EQ.1)GO TO 200
0069                IF(PARAM2.LE.1)GO TO 45
0070                AUX1=PARAM2*2
0071                JCPARM=1
0072                DO 42 I=1,AUX1
0073                AUX2=BITAND(
0074                BITAND(
0075                BITAND(
0076             42 J=J+1
0077             45 IF(PARAM1.LE.16)GO TO 100
0078                J=16-PARAM1

DOS FORTRAN IV 360N-FO-476 2-0          CODIF          DATE  31/13/79      TIME     17.14.51      PAGE 0002

0079                AUX1=PARAM1/2+J-1
0080                J=15
0081                DO 46 I=1,AUX1
0082                BITAND(
0083                BITAND(
0084             46 J=J-1
                    C AHORA ARMO A CODIF DE ACUERDO A SUS BITS
0085            100 CODIF=0
0086                AUX1=1
0087                DO 105 I=1,15
0088                CODIF=CODIF+BITAND(
0089            105 AUX1=AUX1*2
0090                RETURN
0091            200 CODIF=0
0092                RETURN
0093                END
```

3.16.- DESCRIPCION Y FORMATO DE LA FUNCION "CONVTP"

Esta función busca un texto en una tabla de referencia y le regresa su valor.

CONVTP (LITEX, IN, FIN, TABLA, I,J,K)

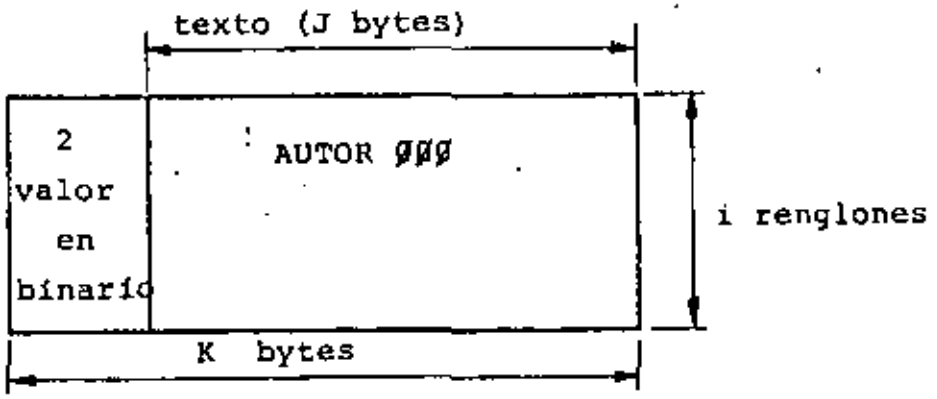
Convierte el texto LITEX (IN → FIN) en un tipo de acuerdo con la tabla TABLA. Nos regresa ese tipo como valor. IN, FIN en bytes.

I indica el número de renglones de la tabla

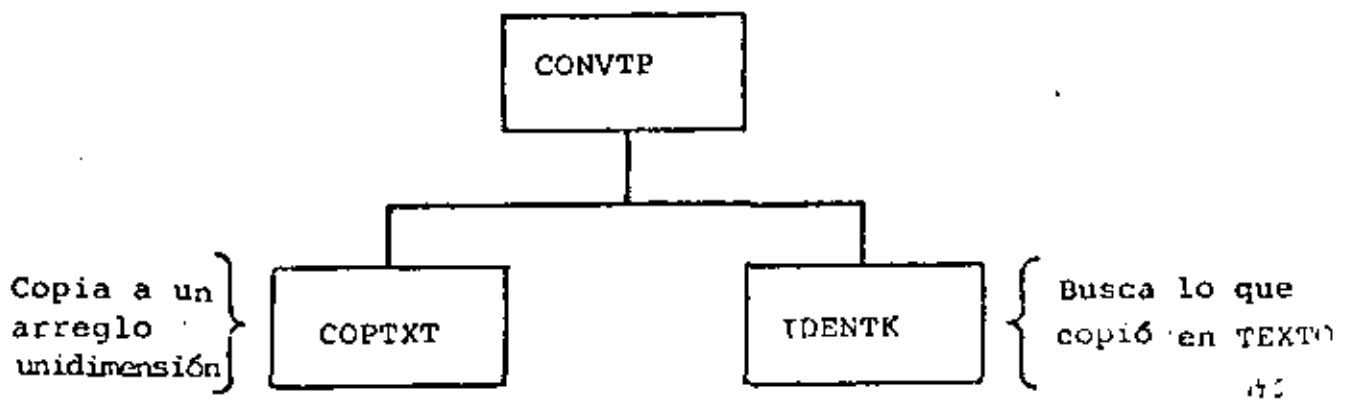
J indica el número de bytes de la parte textual de la tabla, debe de ser par.

K indica número de columnas de la tabla (en dibytes

K = J/2+1



RELACIONES CON OTRAS FUNCIONES Y/O SUBRUTINAS.



3.17.- LISTADO DE PROGRAMA DE LA FUNCION "CONVTP"

```
SOS FORTRAN IV V40N-FD-475 5-8        CONVTP          DATE  07/11/75     TIME  11.25.25      PAGE 0001

0001            INTEGER FUNCTION CONVTP*2(LITEX,IA,FIN,TABLA,II,JJ,FRC
                C     BUSCA EL TEXTO LITEX EN LA TABLA Y ACS DE SU VALOR EN(PRIMERA COLUMNA DE LA
                C     TABLA) REGRESA 0 SI NO LO HALLO JJ DEBE SER PAR, ESTA UN BYTES  IN;FIN BYT
0002            IMPLICIT INTEGER*2 $A-Z@
0003            INTEGER*2 TABLA(7,60),ARR1(200),II/1/,JJ/2/,LITEX(1
0004            LOGICAL TOEXTR
0005            DO 100 I=1,II
0006            CALL COPIX(TABLA,I,I2,JJ/I2,ARR1,II,FRC
0007            IF(NOT.IDENTR(LITEX,IA,FIN,TABLE(1,JJ+1),I2,JJ-1)) GO TO 100
0008            CONVTP=TABLA(1,I@
0009            RETURN
0010      100   CONTINUE
0011            CONVTP=0
0012            RETURN
0013            END
```

63

3.19.- LISTADO DE PROGRAMA DE LA SUBRUTINA "COPTXM"

```
SOS FORTRAN IV V40N-FD-475 5-8        COPTXM          DATE  07/11/75     TIME  11.34.10      PAGE 0001

0001            SUBROUTINE COPTXM(ARR1,I,J,PREGU,K,L,IF)@
                C     COPIA TEXTO DE ARR1 A LA LINEA IF DE PREGU
0002            IMPLICIT INTEGER*2 $A-Z@
0003            INTEGER*2 ARR1(1@),PREGU(1@,3@@
0004            DIMENSION J-I,L-K+1
0005            DO 100 K=1,JTEST
0006            K=K+K-1
0007            CALL DEPS(ARR1(I),ARR1@,I2,PREGU,IFPOS
0008      100   CONTINUE
0009            RETURN
0010            END
```

65

3.18.- DESCRIPCION Y FORMATO DE LA SUBRUTINA "COPTXM"

Esta subrutina lleva a cabo el copiado de un texto de un arreglo a una línea de PREGU.

COPTXM (ARR1, I, J, ARR2, K, L, IF)

Copia texto del arreglo monodimensional ARR1, de los bytes I a J inclusive, al arreglo bidimensional ARR2, a partir del byte K al L inclusve a su fila IF. Solo se llenan los bytes K a L, o menos si i→j es mas corto que K→L.

Estos arreglos contienen dos caracteres/dibyte.

3.20.- DESCRIPCION Y FORMATO DE LA SUBRUTINA "COPTXT"

COPTXT (ARR2D, I, J, L, ARR1DIM, M,N)

Esta subrutina lleva a cabo el translado (copiado) de n bytes de un arreglo a otro.

Copia L dibytes del arreglo ARR2D. (2 dimensiones), al arreglo ARR1DIM (una dimensión), justificado a la izquierda.

I fila de la cual lleva la copia
J a partir de que columna lleva a cabo la copia en dibytes
} ARR2 D

ARR2D es integerX2 de dimensiones (M,N).

Todos los argumentos son en dibytes.

3.21.- LISTADO DE PROGRAMA DE LA SUBRUTINA "COPTXT"

```
CDS FORTRAN IV 26CN-FC-475 5-8      CCPTXT      CATE  07/13/76   TIME  10.11.11   PAGE CCC1
0001            SUBROUTINE COPTXT(ARR2D,I,J,L,ARR1D,M,N)
0002        C   COPIA L BYTES DE ARR2D A ARR1D
0003            IMPLICIT INTEGER*2(A-Z)
0004            INTEGER*2 ARR2D(M,N),ARR1D(1)
0005            DO 100 K=1,L
0006            ARR1D(K)=ARR2D(I,J+K-1)
0007        100 CONTINUE
0008            RETURN
0009            END
```

3.22.- DESCRIPCION Y FORMATO DE LA FUNCION "DAME"

Esta función verifica si el texto TEXTO satisface la pregunta PREGU.

DAME (PREGU, TEXTO) = { T si el texto satisface la pregunta PREGU.
F si no.

TEXTO es un arreglo que contiene un registro lógico para texto de 500 bytes.

ARREGLO INDICS — dónde comienza cada cosa en TEXTO (en dibytes)

			ARREGLO LONGIN — la longitud de cada parte, en dibytes	
1	5	tipo	1
2	6	autor	40
3	46	título	40
4	186	clasificación	·
5	176	idioma	·
6	176	editorial	·
7	256	resumen	·

Una pregunta se lleva a cabo en la siguiente forma:

```
TIPO = LIBRO
     &
AUTOR = JUAN + AUTOR/RODRI (/ = quiere decir contenga)
TITULO/NIN + TITULO/CHILD
           &
IDIOMA = ESPAÑOL + IDIOMA = INGLES
           &
EDITORIAL = DIANA
           &
```

RESUMEN/MEXIC

&

CLASIFICACION/PSICO

&

TABTPX: Tabla de conversión donde:

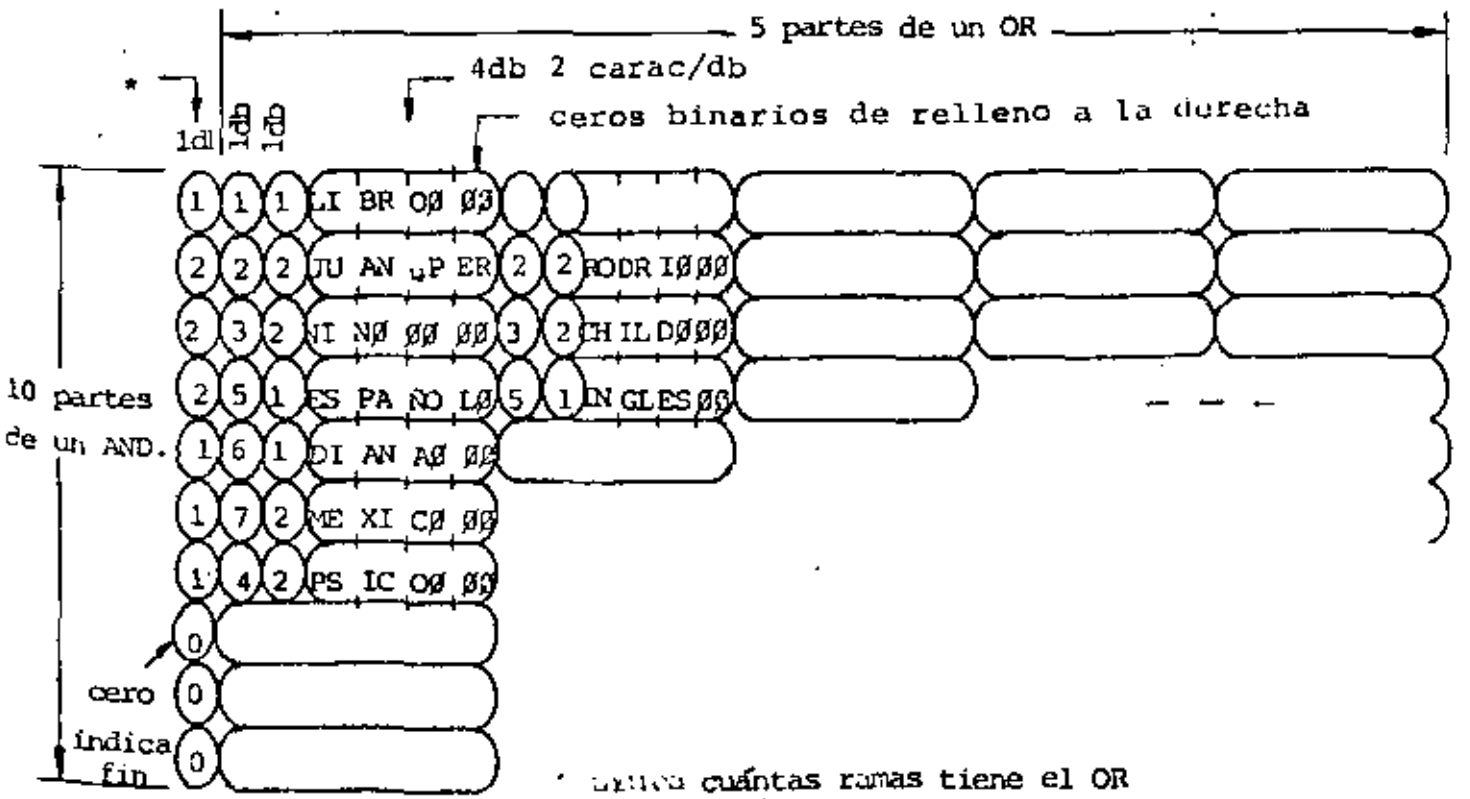
= - 1 igual
/ - 2 contiene

TIPO -1:

- Libro	- 1
- Revista	- 2
- Otro	- 3
- Comentario	- 4
Autor	- 2
Título	- 3
Clasificación	- 4
Idioma	- 5
Editorial	- 6
Resumen	- 7

|— 10 caract. —|

Así se guarda en el arreglo PREGU.



3.23.- LISTADO DE PROGRAMA DE LA FUNCION "DAME"

```
DOS FORTRAN IV 360N-FO-475 3-F        DAME          DATE  07/11/15     TIME     11.20.26      PAGE 0001

0001            LOGICAL FUNCTION DAME*8(PREGL,TEXTO)
                C     DAME .T. SI EL TEXTO SATISFACE LA PREGUNTA
0002            1=PLEIEIT INTEGER*2 IA-Za
0003            INTEGER*2 PREGUNTO,31m,TEXTO31m,INDICES37m/7,11,91,171,181,201,2637
0004            INTEGER*2 LOGIN81m/2,80,80,10,20,60,234/,PALABR844
0005            INTEGER*2 TATTTETEO/ALTA,OPPOTAL ATA  A.ASEGIAVIA.OSTAIA/A.OCTA/,
                1ARCA,A  A,A  A,ACCO,AMEA,ANIA,ASPA/
0006            LOGICAL EXISTE
0007            DATA T1,18,137/1*8,32/
0008            DAME=.TRUE.
0009            DO 200 I=1,10
0010            IF(PREGL(I).EQ.LOMPIENA
0011            PREGL(I)=1m
0012            DO 100 J=1,F
0013            MA=2**J-1D
0014            DO 300 KA=1,4
0015      300   PALABR(KA)=PREGL(I,K+1m
0016            KK=PREGL(I)/2**(J-1)*2m
0017            IF(K.EQ.0)GO TO 100
0018            IF(K.NE.1)GO TO 70
0019            IF(.NOT.EXISTE(PALABR,I1,I8=AUCERC(PALABR,18I,TEXTO,INDICES(KK),
                1GO TO 100
0020            KK=KK+INDICES(KK+1m/2
0021            IF(INDICES(KK+1).NE.2)IF(I8=I1)GO TO 300
0022            GO TO 200
0023       70   IF(.NOT.EXISTE(PALABR,I1,I8=AUCERC(PALABR,I8I,TEXTO,INDICES(KK),
                1INDICES(KK+1)=LOGIN(KK+1)))GO TO 100
0024            GO TO 200
0025      300   CONTINUE
0026            DAME=.FALSE.
0027            RETURN
0028      200   CONTINUE
0029            RETURN
0030            END
```

70

3.24.- DESCRIPCION Y FORMATO DE LA SUBRUTINA "DESPOL"

Esta subrutina despolariza los años.

DESPOL (V, V1, V2)

Despolariza un valor V de la forma 148.235

en los valores V1 = 148

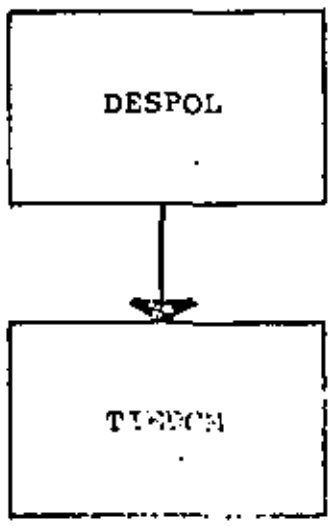
V2 = 235

Si V2 = 0, entonces se hace V2 = V1

V es real; V1 y V2 son integer X 2

Acepta hasta 4 decimales de V, los demas cienmilésimas, millonésimas, etc, son ignorados y los toma como si fueran ceros.

Relación con otras subrutinas y/o funciones.



Verifica si un número tiene o no parte decimal.

3.25.- LISTADO DE PROGRAMA DE LA SUBRUTINA "DESPOL"

```
EOS FORTRAN IV 3604-FO-415 3-1        DESPOL        DATE  07/13/75      TIME    11.31.47      PAGE 0001

0001          SUBROUTINE DESPOL(V,N1,V2)
0002          IMPLICIT INTEGER(A-Z)
0003          REAL V,DECI
0004          LOGICAL TIEDO
0005          VERIFIX=0
0006          DECI=V-N1
0007          DO 100 J=1,4
0008          IF(.NOT.TIEDO*DECI.GE.1.OR.DECI.LE.0) GO TO 200
0009          DECI=DECI*10.
0010      100 CONTINUE
0011      200 CONTINUE
0012          IF(DECI*.GT.5.OR.DECI.EQ.5.AND.DECI*10-INT(DECI*10).GT.0) V
0013          VERIFIX=VERIFIX+1
0014          IF(V1.EQ.0) V1=1
0015          RETURN
0016          END
```

72

3.26.- DESCRIPCION Y FORMATO DE LA SUBRUTINA "DEPST2"

73

Esta subrutina su función es depositar byte por byte de un arreglo a otro.

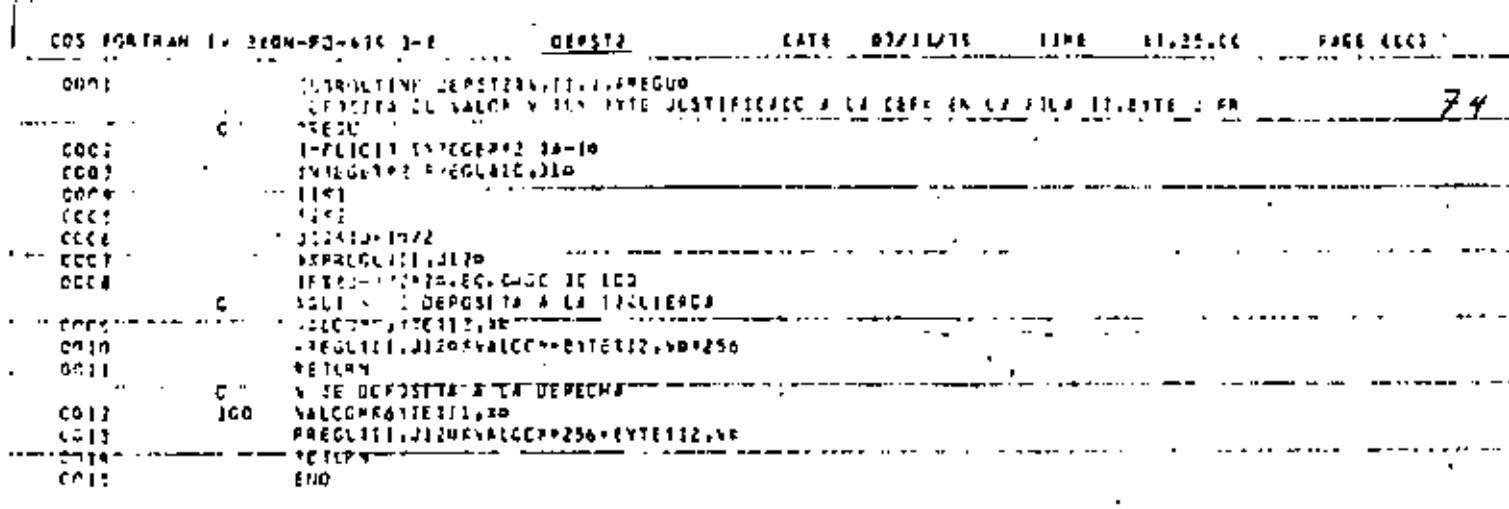
DEPST2 (V, II, J, PREGU)

Deposita el valor V en la fila II, byte J del arreglo bidimensional PREGU.

no altera nada mas.

V es un byte justificado a la derecha.

3.27.- LISTADO DE PROGRAMA DE LA SUBRUTINA "DEPST2"



```
COS FORTRAN (V 2.0ON-PD-81C )-E        DEPST2          DATE  82/11/15      TIME  11.25.CC      PAGE 0001
0001                SUBROUTINE DEPST2(A,II,J,PREGUN
                    C DEPOSITA EL SALDO Y SUS BYTE JUSTIFICADO A LA DER. EN LA FILA II,BYTE J EN
                C   *RECIO
0002                INTEGER*2 A(10)
0003                BYTE BYTEGR*2 EQUIVALENCE(I
0004                III=1
0005                JJJ=2
0006                JJJ=JJJ+1972
0007                BYTEGR(III,JJJ)=0
0008                IF(BYTEGR(III,JJJ).EQ.CHAR(0)) GO TO 100
                C   SOLO SI DEPOSITA A LA DERECHA
0009                CALL MVBITS(A(1),0,8,BYTE12,0)
0010                BYTEGR(III,JJJ+A(1)*256)=BYTE12
0011                *RETURN
                C   SI SE DEPOSITA A LA DERECHA
0012            100 BYTEGR(III,JJJ)=A(1)
0013                BYTEGR(III,JJJ+A(1)*256)=BYTE12
0014                *RETURN
0015                END
```

3.28.- DESCRIPCION Y FORMATO DE LA FUNCION "DIME"

DIME (PREVAL)

Esta función es verdadera (T) si se satisface el predicado PREVAL; si no es falsa (F).

El predicado que está guardado en el arreglo PREVAL se aplica al ítem que ya está guardado en memoria con las restricciones dadas por IAÑO, IENT, IEDAD, ISEXO. Todos estos parámetros se pasan por área COMMON.

(Ver formato de PREVAL que se encuentra en la función BUSCAI (Interactivo).

3.30.- DESCRIPCION Y FORMATO DE LA FUNCION "EGUAVAL"
(Encuestas)

Similar a EVAL (encuestas), pero guarda el valor VALOR en el lugar indicado.

EGUAVAL (# cuadro, i, j, k, l, m, n, ii, jj, kk, ll, mm, nn, VALOR)

EGUAVAL trabaja con el registro lógico del cuadro de la encuesta ya en memoria. No escribe en el disco. Solo escribe en memoria.

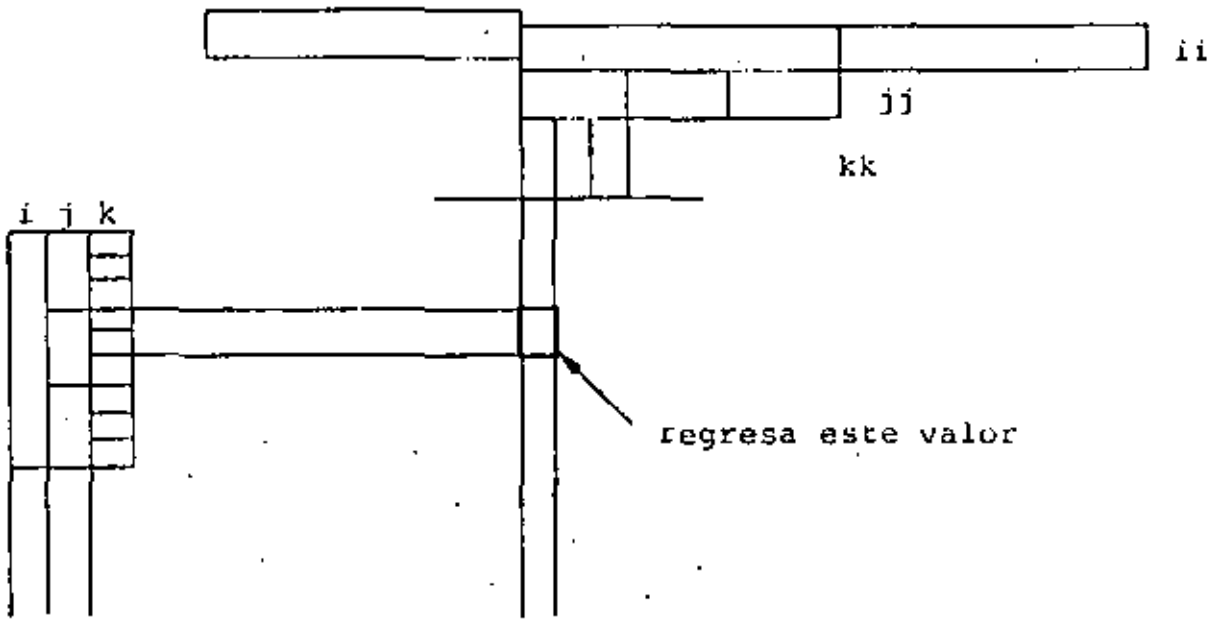

```
0001            SUBROUTINE EGUAVAL(E,V,D,Z,*)
0002            IMPLICIT INTEGER*2 (A-Z)
0003            WRITE(3,100)N,E,V
0004            IF(A.EQ.0)CALL EXIT
0005            RETURN 1
0006      100   FORMAT(1X,'*****ERROR*****',I4,1X,I4,' VALOR ',I4)
0007            END
```



```
0001            LOGICAL FUNCTION EXISTEPALABR,LFI,LFF,TEXTO,IA,IFIN,IFA
     C          BUSCA EN TEXTO DESDE IA HASTA IFIN LA PALABR (LFI---LFF
0002            IMPLICIT INTEGER*2 (A-Z)
0003            LOGICAL IDENTK
0004            INTEGER*2 PALABR(10),TEXTO(10)
0005            EXISTE=.TRUE.
0006            IF(LFI.GT.LFF)RETURN
0007            DO 1 I=IA,IFIN
0008            IF(IDENTK(PALABR,LFI,LFF,TEXTO,I,IFIN))RETURN
0009      1     CONTINUE
0010            EXISTE=.FALSE.
0011            RETURN
0012            END
```

3.36- DESCRIPCION Y FORMATO DE LA FUNCION "EVAL" (Encuestas)

Esta función regresa el valor ... cuado de un cierto cruce de fila y renglón en un cuadro dado.



EVAL (# de cuadro, I,J,K,L,M,N, , II,JJ,KK,LL,MM,NN)

I,J,K,L,M,N: índices horizontales (poner cero en los que sobren)

II,JJ,KK,LL,MM,NN: índices verticales (poner cero en los que sobren)

EVAL trabaja con el registro lógico del cuadro de las encuestas ya en memoria.

EVAL regresa el valor que el cuadro arroja.

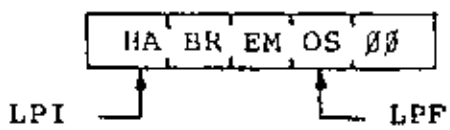
DESCRIPCION Y FORMATO DE LA FUNCION "EXISTE

EXISTE (PALABR, LPI, LPF, TEXTO, IN, IFIN, IP) { T si palabra E Texto / F si no.

La función es verdadero (T), si la PALABR pertenece al TEXTO, y falso (F) en caso contrario.

PALABR. Arreglo monodimensional con los bytes LPI a LPF, llenos de texto, 2 caract/dibyte.

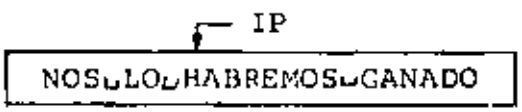
Ejemplo:



TEXTO. Arreglo monodimensional con caracteres desde IN (# de byte) hasta IFIN (# de byte) inclusive.

EXISTE. busca en TEXTO desde IN hasta IFIN ambos inclusive, la parte de PALABR (LPI LPF), ambos inclusive.

Regresa T si lo encontró. Además, deja a IP (lo modifica) apuntando a la primera letra (al byte) donde está PALABR en TEXTO, o sea, deja a IP apuntando como se indica:



- Todos los apuntadores de EXISTE (de sus argumentos) son bytes. Apuntan a bytes.

- Los ceros binarios de PALABR casan con lo que sea y hacen que la búsqueda se satisfaga.

- Los caracteres de PALABR deben estar cargados a la izquierda. Desde el byte 1.

- (No debe haber un cero binario en el byte 1).

- Deja a IP apuntando a IFIN+1 (byte) si no existe.

3.38.- DESCRIPCION Y FORMATO DE LA FUNCION "GUAVAL" 83

GUAVAL (NPROP, VAL, SEXO, POS)

Esta función lógica, guarda el valor VAL, de la propiedad NPROP, según el sexo SEXO, en la edad que empieza en la posición POS. (NPROP no debe de ser una propiedad computada).

GUAVAL regresa a <u>verdadero</u> si se pudo guardar el valor y <u>falso</u> si no.

El valor VAL que se guarda puede ser:

- Ya sea en la zona por posición.
- O en la zona ordenada si ya se encontraba ahí NPROP.
- O en la zona de propiedades desordenadas.
- Incrementa o altera la cabeza del registro.
- No lleva a cabo la copia el registro a disco.

```
0045          100   GLAVALE,TRUE.
0046                RETURN GLAVALE
0047                END-IF
0048                RETURN
0049          111   VENTECDIS(ICESAL...
0050                VENTECDIS(ICE-IC
 051                RETURN
                C NO HUBO FORMA DE CUADRAR EL VALOR, REGRESO GLAVAL FALSO.
0052          333   GLAVALE,FALSE.
0053                RETURN
0054                END
```

3.40.- DESCRIPCION Y FORMATO DE LA FUNCION "IDENTK"

IDENTK (PALABR, LPI, LPF, TEXTO, I, IFIN)

IFIN es el parámetro que indica final de texto (byte)

LPI, LPF, es el parámetro que indica los bytes inicial y final de PALABR, donde PALABR es un arreglo integerX2, empacado con 2 caracteres/dibyte. Los ceros binarios de PALABR casan con cualquier carácter.

3.41.- LISTADO DE PROGRAMA DE LA FUNCION "IDENTK"


```
ECS FORTRAN IV (ECN-PC-V15 V-4          IDENTK          CCCC  03/11/75     LPPF    11.32.41      PAGE CCC1

0001                 LOGICAL FUNCTION IDENTK(PALABR,LPI,LPF,TEXTO,II,IFIN)
                     *    ESTA DE LAS PALABRAS (LPI--LPF) Y EL TEXTO (II--IFIN) SON IDENTICOS
0002                 IMPLICIT INTEGER*2 (A-Z)
0003                 INTEGER*2 PALABR(10),TEXTO(1)
0004                 IDENTK=.FALSE.
0005                 DO 1 I=LPI,LPF
0006                 APOSTE(I,PALABR)
0007                 IF(ALOG(TEXTO(II+I-LPI)
0008                 KRIT=I-LPI
0009                 IF(KRIT.LT.IFIN-II+1)GO
0010                 IF(A.NE.B)GO TO 2
0011             1   CONTINUE
0012             2   IDENTK=.TRUE.
0013                 RETURN
0014                 END
```

3.42.- DESCRIPCION Y FORMATO DE LA FUNCION "LANOBL"

LANOBL (TEXTO, IN, IF)

Esta funci[illegible] la última posición del carácter no blanco (que no sea un espacio en blanco) de derecha a izquierda del texto TEXTO, que se encuentra entre los límites IN e IF. TEXTO es un arreglo integer X 2 de caracteres, dos por cada dibyte.

Regresa 0 si no hay caracteres no blancos.

3.43.- LISTADO DE PROGRAMA DE LA FUNCION "LANOBL"


```
FCS FORTRAN IV [CDC-180-875 3-0        LANOBL          DATE  07/11/75     TIME  11.34.17     PAGE 0001
0001                  SUBROUTINE LANOBL(TEXTO,IN,IF)
                      C     ... ULTIMA POSICION DEL TEXTO, NO BLANCO DE TEXTO
                      IMPLICIT INTEGER(A-Z)
0003                  DIMENSION TEXTO(2),BLANCO(2)
0004                  DATA BLANCO/2H  /
0005                  EQUIVALENCE(IC,BLANCO)
0006                  DO 100 I=IN,IF
0007                  IF(TEXTO(I).NE.IC) GO TO 200
0008                  CONTINUE
0009                  IF(TEXTO(I).NE.IC) GO TO 200
0010                  CALL ERRSET(115,0,0,0,0,0)
                      RETURN
0012                  IF=0
0013                  RETURN
                      END
```

3.44.- DESCRIPCION Y FORMATO DE LA FUNCION "LENTRE"

LENTRE (A, B, C)

Esta funcion lógica es verdadera (T) si B se encuentra entre A y C es decir:

$$A \leq B \leq C$$

Todos sus argumentos son enteros de 2 bytes.

3.45.- LISTADO DE PROGRAMA DE LA FUNCION "LENTRE"

```
COS FORTRAN V5 2.0N-7.0-425 5-E      LENTRE      DATE 07/31/75   TIME 11.26.27   PAGE 0001
0001          LOGICAL FUNCTION LENTRE(A,B,C)
0002     C    TRUE SI A.LE.B.LE.C.  C SEA E ESTA ENTRE A Y C, ENTEROS DE 2 BYTES
0002          IMPLICIT INTEGER*2 (A-Z)
0003          LENTRE=A.LE.B.AND.B.LE.C
0004          RETURN
0005          END
```

3.46.- DESCRIPCION Y FORMATO DE LA FUNCION "LEVARI"

LEVARI(AÑO, ENTID, EDAD, SEXO)

Todos los argumentos son de salida. Lee del teclado.

Lee AÑO en este formato:

1975
75
1975.77

Y los pasa (los saca por sus argumentos) respectivamente así:

1975.0
1975.0
1975.77

Lee ENTID (Entidad) en este formato:

AGUASCALIENTES

Y lo pasa a:

00001 (del apéndice "A" código de propiedades).

Lee SEXO en este formato:

M	1
F	2
A	0
AMBOS	0
TODOS	0
1	1
2	2
0	0

NOTA: Esto es idéntico a lo que usa la función BUSCA (Batch).

Lee EDAD en este formato:

1

3

3.5

Y los pasa a:

1

3

3.5

Todos son argumentos REALES (flotante), verifica lo que leyó y regresa argumentos legítimos.

3.49.- LISTADO DE PROGRAMA DE LA SUBRUTINA "LLELIN"

```
DOS FORTRAN IV 360N-FO-475 3-6        LLELIN          DATE  03/11/75    TIME  11.23.27      PAGE 0001

0001                  SUBROUTINE LLELIN(LETRA,EL,FRECU,I6
                C     LLENA DE CEROS LA LINEA 1 DEL ARREGLO EL(1)=ENSIGNAL FRECU
0002                  IMPLICIT INTEGER*2 (A-Z)
0003                  INTEGER*2 LABTPA(7,60/1,2,3,4,5,6,7,8,10,45U6,8T16,8CL8,8IC8,8EC8,
                     1T8L8,8P8,8IC8,8TL8,8A58,8IC8,8I18,8SU8,8   8,8   8,8IC8,8I18,8MA8,
                     2T8  8/
0004                  INTEGER*2 LITCRT16,PREGU$17,310,FLEEL$306,EL/8   8/,MAS/4H  /
0005                  INTEGER*2 IGUAL/6HAS4 /;DIAG/8H   /;FCNTE/8   8/
0006                  LOGICAL EXISTE
                C     FLEBLA ES= LAS FLECHAS BLANCAS
0007                  DATA IIII,II,II,I7,IIC/1,7*6,7*32/
0008                  PUNTIF@(1)=2,FLAIC0
0009                  DIAG(80)=IGUAL,DIAG
0010                  PREPRO@LLELIN(LETRA,IIII,LLO
0011                  IF(80)=PP,LITE20,EC,FLAIC)GO TO 400
0012            10C   FLEBLA(1)=FF+1
                      DO 105 IFLEL=2,9
0013                  IIFLDLE(IFLEL)
0014                  IF(.NOT.EXISTE(MAS,IIII,IIII,LITE),IIII)FLEEL(1)(IFLEL)=IC,II,IP66
                      GO TO 110
0015                  FLEBLA(1)(IFLBLA)=IF
0016            105   CONTINUE
0017                  IIFLDLE=6
0018            110   FLEBLA(1)(FLBLA)=BLANCO(LITE),IIII,CLE(1)
0019            200   IIPIR(IIFLDL-1)
                      DO 205 I=1,IIPI
                C     REVISA TEATO ENTRE DOS DE FLECHAS BLANCAS
0020                  IF(EXISTE(IGUAL,IIII,IIII,LITE),FLEEL(1)(,FLEEL(2)(,IIII)G,IP66
                      GO TO 210
0021                  IF(EXISTE(DIAG,IIII,IIII,LITE),FLEEL(1,0,FLEEL(2)(,IIII)G,IP66
                      GO TO 210
0022                  WRITE(3,10000
0023            10000 FORMAT(1H  NO HUBO A MI /60
0024            210   FLEPLA=I
0025                  PREGU(I)=FRECU(I)
0026                  PREGU(I,EQ(CONTIFLEL(LA),FRACBL(LITE),IIII)FLEEL(2)(,FLEPLA-IIII),
                     1LANOBL(LITE),IIII)=FLEBLA(2)(,FLEPLA-IIII),T(ETRA,I7,IIC,I66
0027                  GO TO 1
0028                  PREGU(I,EQ(1
0029                  IF(80)=FLEBLA,LITE20,EC,CIAG)=PREGU(I),EQ(2
0030                  GO TO(3,4),IOPI2
0031                  I=REC)+1
0032                  CALL COPY(PIXITE),FRACBL(LITE),FLEBLA(2)(,FLEEL(2)(,IIII)-IIII),
                     1LANOBL(LITE),FLEPLA(IIII,FLEEL(2)(IIII)-IIII),PRIEC,I7,I16,I6
0033            205   CONTINUE
0034                  PREGU(I,IO)=IFLBL-1
0035                  RETURN
0036            400   PREGU(I,IO)=0
0037                  RETURN
0038                  END
```

3.48.- DESCRIPCION Y FORMATO DE LA SUBRUTINA "LLELIN"

LLELIN (LITEX, L, PREGU, i)

Donde:

LITEX: Es un arreglo monodimensional, que contiene caracteres, uno en cada 8 bits.

L : Es la longitud en bytes de LITEX.

PREGU: Es el arreglo de la pregunta.

i : Es el índice, por medio del cual la subrutina LLELIN, llena la línea i del arreglo PREGU.

Ejemplo: AUTOR = JUAN PERES + AUTOR/RODRI

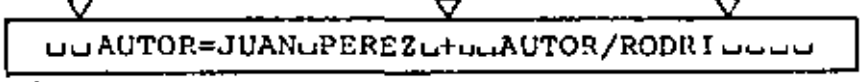
Forma en que trabaja:

En la línea i del texto LITEX

0).- Si el primer carácter es un ., salta pone un cero en (i,1) y fin

1.- Pone una marca en los puntos no blancos y en cada signo +

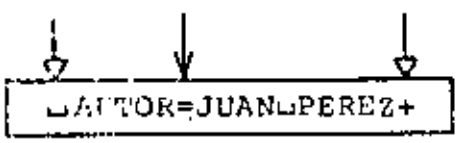
por ejemplo:



2.- Para cada texto entra un par de

a) Para signos (=,/) lo busca y cuando lo encuentre le pone

ejemplo:



b) Lo que está a la izquierda de ↓ lo codifica en un número del 1 al 7 de la tabla de conversión que se encuentra en la función BUSCAI.

Por ejemplo, AUTOR se convierte en 2, usa EXISTE para llevar a cabo esta conversión.

Si entre ▽ y ↓ existe TIPO, poner 1
Si entre ▽ y ↓ existe AUTOR, poner 2 etc.

NOTA: Si lo que está a la izquierda de ↓ es un carácter numérico, transformarlo a binario y no pasar por tabla de conversión.

El número de la tabla que se determine en (i,2+(j-1)*6) de PREGU.

c) Lo que está debajo de ↓ se mete en (i,3+)j-1)*6) de PREGU

Si es (=) pone un 1
Si es (/) pone un 2

d) Lo que está a la derecha de ↓ lo mete en j de PREGU en la línea i de (1,4+(j-1)*6) a (i,7+(j-1)*6)

e) Aumenta el contador de ramas del OR.

3.- Guarda el contador de 2 e).- en (i,1)

3.50- DESCRIPCION Y FORMATO DE LA FUNCION "NUCERO"

NUCERO (ARR,N)

Esta función da el número de bytes llenos de ceros binarios a la derecha del arreglo ARR, el cual es de N bytes de largo.

ARR es integerX2

3.51.- LISTADO DE PROGRAMA DE LA FUNCION "NUCERO"


```
DOS FORTRAN IV 2ECN-FC-415 3-E        NUCERO          DATE  03/11/25    TIME    11.36.41      PAGE 0001
0001                  INTEGER FUNCTION NUCERO*2(ARR,N)
                C     * ESTA FUNCION DA EL NUMERO DE BYTES LLENOS DE CEROS A LA DERECHA DE ARR
0002                  IMPLICIT INTEGER*2 (A-Z)
0003                  INTEGER*2 ARR(1)
0004                  DO 100 I=1,N
0005                  I=N-I
0006                  IF(ARR(I).EQ.0)GO TO 100
0007                  NUCERO=I-1
0008                  RETURN
0009          100     CONTINUE
0010                  NUCERO=N
0011                  RETURN
0012                  END
```

3.52.- DESCRIPCION Y FORMATO DE LA FUNCION "NUMERC"

NUMERC (EEE, I, D)

Esta función es verdadera (T) si desde I hasta D hay puros números (caracteres enteros del 0 al 9) en el arreglo EEE, el que debe de ser empacado a 2 caracteres por dibyte.

I, D apuntan a bytes.

3.55.-LISTADO DE PROGRAMA DE LA FUNCION "PRNOBL"


```
EOS FORTRAN IV 360H-FD-435 3-6      PRNOBL      DATE  07/11/75      TIME  11.33.57      PAGE 0001

0001          INTEGER FUNCTION PRNOBL*2(TEXTO,IN,IFIN)
              C     NOS DA LA POSICION DEL PRIMER BYTE NO BLANCO DE IZQUIERDA A DERECHA
0002          IMPLICIT INTEGER*2 (A-Z)
0003          INTEGER*2 TEXTO(10),BLANCO/'  '/,I2/2/
0004          DATA I0,I1,I5/0,1,5/
0005          BLANTO=TEXTO(2,BLANCO
0006          DO 100 PRNOBL=IN,IFIN
0007          IF(BYTE(TEXTO,PRNOBL).NE.BLANTO)RETURN
0008      100 CONTINUE
0009          CALL ERRORS(I1,I0,I0,I1,I5)
0010          PRNOBL=0
0011      110 RETURN
0012          END
```

PRO (NPROP, RELACI, NUM)

Este programa es una función lógica, que relaciona un valor de propiedad o de dato de encuesta con el valor dado por el usuario. Las relaciones lógicas son "mayor que", "menor que", "igual a", "mayor o igual a", "menor o igual a", "diferente de". La respuesta es "cierto" si el valor de la propiedad NPROP está en la relación RELACI con el número NUM.

3.58.- DESCRIPCION Y FORMATO DE LA FUNCION "PROP" 101

PROP (NPROP, RELACI, N1, N2)

Esta función lógica relaciona el valor de la propiedad o de datos de encuesta con un intervalo de valores dados por el usuario N1 y N2, la respuesta es cierta sí el valor de la propiedad NPROP está en la relación RELACI en el intervalo N1 a N2, y la respuesta es falsa si no se cumplen las condiciones anteriores.

3.54.-DESCRIPCION Y FORMATO DE LA FUNCION "PRNOBL" 102

Esta función da la posición del primer carácter no blanco en el vector de búsqueda.

PRNOBL (TEXTO, IN, IF)

Da la primera posición (byte) no blanca (que no sea un espacio en blanco) de izquierda a derecha del texto TEXTO, que se encuentra en los bytes IN (Inicial) e IF (Final, ambos inclusive).

TEXTO es un arreglo integerX2 de caracteres, dos por cada dibyte. Regresa o si no hay caracteres no-blancos.

3.60.- DESCRIPCION Y FORMATO DE LA FUNCION "REGEN"

REGEN (ARCH, PARAM, PARAM, ..., ARCHSA, E)

Regresa el archivo ARCH (archivo tipo DIF.78) y lo guarda en ARCHSA.

Es decir, copia ARCH hacia el (nuevo) archivo ARCHSA, el cual tiene ciertos parámetros PARAM, PARAM,.....

- No destruye ARCH.
- Rehace el directorio. (el de salida, el de ARCHSA).
- Toma las propiedades desordenadas y las mete en el lugar que les toque en el área de propiedades ordenadas.
- Compacta (o expande) los registros lógicos de ARCH de acuerdo con los parámetros que REGEN indica:

REGEN = { T si todo salió bien
        { F si algo salió mal. En este caso, E recibe un número que es el número de error, número del mensaje de error.

- No es para el usuario.
- Si ARCH es 0, genera un nuevo banco, <u>limpio.</u>

3.62.- DESCRIPCION Y FORMATO DE LA FUNCION "RELAC"

RELAC (NR, VAL, N, M)

La función es verdadera (T) si VAL está en la relación NR con N, si no es falso (F).

NR, es el número de relación lógica, de la tabla que se encuentra en la funcion BUSCAI (Interactivo)

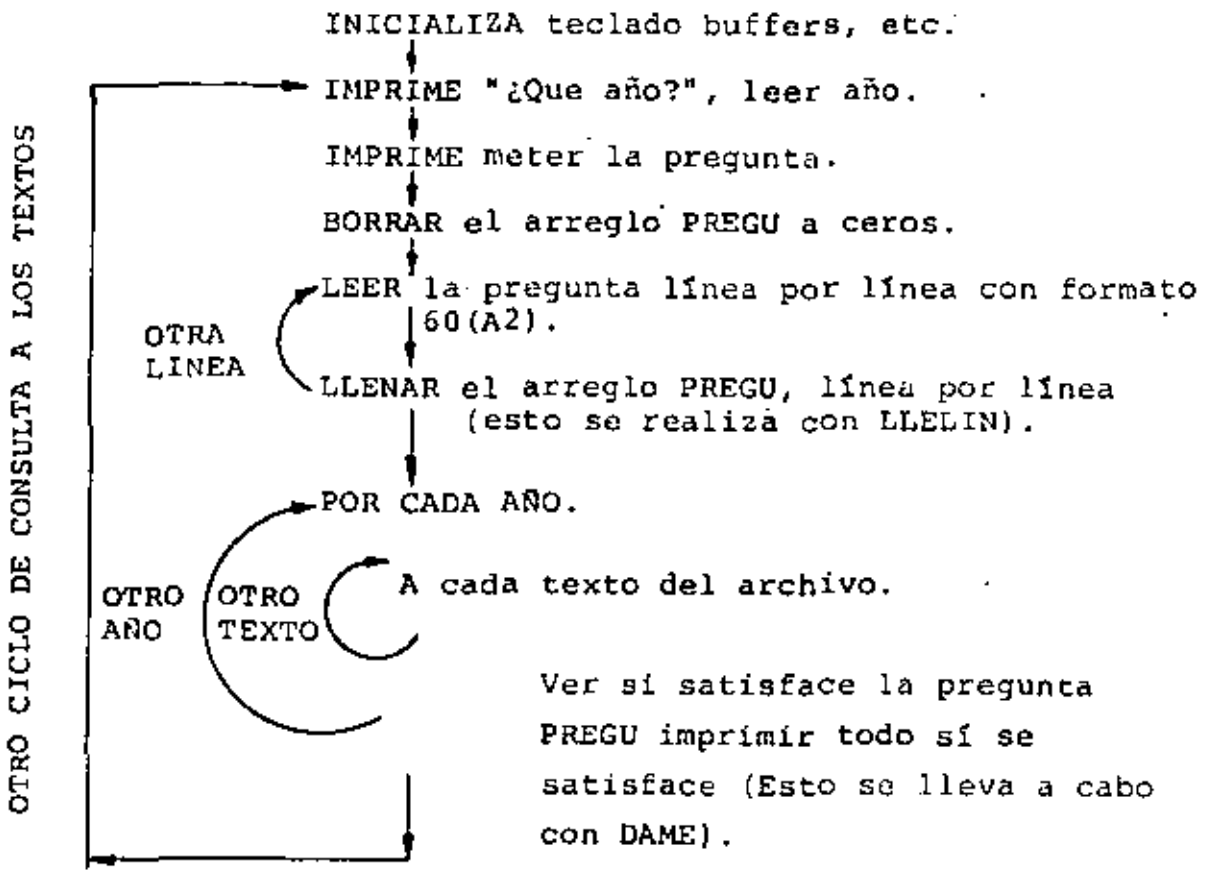
Si NR = 10 ("entre"), entonces VAL debe de estar en N y M, por lo que $N \leq VAL \leq M$

3.64.- DESCRIPCION Y FORMATO DE LA FUNCION "TEXTOS" (Interactivo)

1.- Este es un programa principal que desde el teclado de entrada, lee textos en el formato de TEXTO, e imprime todos los textos que satisfagan la pregunta hecha ya sea impreso en papel o por pantalla.

2.- Se regresa otra vez a 1.

Diagrama de flujo de TEXTOS (Interactivo).



Al terminar de imprimir se regresa.

3.66.- EJEMPLO DE CONSULTA DE TEXTO


```
CLAVE PARA CONSULTAS

HACER LA PRECLASE

     25    75

NUMERO DE TEXTO     1
CLASE DE TEXTO  1
AUTOR SILVA LINDBERG
TITULO COMPUTERS IN LIFE SCIENCE RESEARCH
CLASIFICACION 574.5
IDIOMA INGLES
EDITORIAL FASES MONOGRAPHS
FECHA EDICION
CLASIFICACION MEDICINA BIOLOGIA APLICACIONES


NUMERO DE TEXTO     2
CLASE DE TEXTO  1
AUTOR GEORGE E.P. BOX & GWILYM M. JENKINS
TITULO TIME SERIES ANALYSIS
CLASIFICACION 519.5
IDIOMA INGLES
EDITORIAL HOLDEN-DAY
FECHA EDICION
MATEMATICAS ESTADISTICA


NUMERO DE TEXTO     4
CLASE DE TEXTO  1
AUTOR B.P. DENNISOVICH & T.A. PARCH
TITULO COMPUTATIONAL MATHEMATICS
CLASIFICACION 519.4
IDIOMA INGLES
EDITORIAL MIR
FECHA EDICION
COMPUTACION ALGORITMOS MATEMATICOS
```

EJEMPLO DE CONSULTA DE TEXTO

```
CLE APRENDIZAJE CONSULTAR

HACER LA PREGUNTA                                                    107

        75      75

NUMERO DE TEXTO      1
CLASE DE TEXTO  1
AUTOR SILIA LINDBERG
TITULO COMPUTERIZED LIFE SCIENCE RESEARCH
CLASIFICACION 574.0
IDIOMA INGLES
EDITORIAL PASES MENOGRAFICOS
FECHA 751074
TEMAS CIENCIA MEDICINA BIOLOGIA APLICACIONES


NUMERO DE TEXTO      2
CLASE DE TEXTO  1
AUTOR D.P. DENDOVICH & T.A. PARCH
TITULO COMPUTATIONAL MATHEMATICS
CLASIFICACION 105.A
IDIOMA INGLES
EDITORIAL MIR
FECHA 010274
COMPUTACION ALGORITMOS MATEMATICAS


NUMERO DE TEXTO      3
CLASE DE TEXTO  1
AUTOR ROBERT R. KORFHAGE
TITULO LOGICA Y ALGORITMOS
CLASIFICACION 423.*
IDIOMA ESPAÑOL
EDITORIAL LIMUSA
FECHA 140774
INFORMATICA COMPUTACION MATEMATICAS


NUMERO DE TEXTO      4
CLASE DE TEXTO  1
AUTOR ARECHIGA, CORDERO, DOMINGUEZ, GONZALEZ
TITULO FUNDAMENTOS DE COMPUTACION
CLASIFICACION  31.A
IDIOMA ESPAÑOL
EDITORIAL CONSEJO EDITORIAL POLITECNICO
FECHA 041174
HISTORIA, EVOLUCION, LOGICA, CODIGOS, ARQUITECTURA DE LA COMPUTADORA, DIAGRAMAS
DE FLUJO, PROGRAMACION, APLICACION
```

EJEMPLO DE CONSULTA DE TEXTO

QUE MAS DESEA CONSULTAR

HACER LA PREGUNTA

TS TS

NUMERO DE TEXTO 3
CLASE DE TEXTO 1
AUTOR MURRAY R. SPIEGEL
TITULO ESTADISTICA
CLASIFICACION 330.4AB
IDIOMA ESPAÑOL
EDITORIAL MCGRAW-HILL
FECHA 200110
TEORIA PROBLEMAS RESUELTOS MATEMATICAS ESTADISTICA

NUMERO DE TEXTO 4
CLASE DE TEXTO 1
AUTOR B.P. DEMIDOVICH & I.A. MARON
TITULO COMPUTATIONAL MATHEMATICS
CLASIFICACION 519.4
IDIOMA INGLES
EDITORIAL MIR
FECHA 010316
COMPUTACION ALGORITMOS MATEMATICOS

EJEMPLO DE CONSULTA DE TEXTO

QUE MAS DESEA CONSULTAR

HACER LA PREGUNTA

TS TS

NUMERO DE TEXTO 3
CLASE DE TEXTO 1
AUTOR MURRAY R. SPIEGEL
TITULO ESTADISTICA
CLASIFICACION 330.4AB
IDIOMA ESPAÑOL
EDITORIAL MCGRAW-HILL
FECHA 200110
TEORIA PROBLEMAS RESUELTOS MATEMATICAS ESTADISTICA

NUMERO DE TEXTO 4
CLASE DE TEXTO 1
AUTOR B.P. DEMIDOVICH & I.A. MARON
TITULO COMPUTATIONAL MATHEMATICS
CLASIFICACION 519.4
IDIOMA INGLES
EDITORIAL MIR
FECHA 010316
COMPUTACION ALGORITMOS MATEMATICOS

3.67.- DESCRIPCION Y FORMATO DE LA FUNCION "TIEDCM"

TIEDCM(V)

Esta funcion es verdadera (T) si V tiene decimales a la derecha del punto decimal, por ejemplo: 3.1416; y es falso (F) si a la derecha del punto decima. hay ceros, por ejemplo: 1970.0

V es un argumento real.

DESCRIPCION Y FORMATO DE LA FUNCION "TRACAN"

TRACAN (EEE, I, D)

Esta funcion transfiere los caracteres numéricos (para poder llegar los caracteres a este paso fué necesario que primero pasaran por una prueba de verificación) desde I hasta D en EEE a un valor numérico, el que regresa EEE se empaca a 2 caracteres/dibyte.

I, D apuntan a bytes.

3.68.- LISTADO DE PROGRAMA DE LA FUNCION "TIEDCM"

```
IBM FORTRAN IV 2400-FO-415 5-8        TIEDCM        DATE  07/31/75      TIME  11.21.45      PAGE 0001
0001          LOGICAL FUNCTION TIEDCM*10
      C       TIPO SI Y SOLO SI TIENE DECIMALES DE CERO
0002          IMPLICIT INTEGER*2 (A-Z)
0003          REAL *4
0004          TIEDCM=FLOAT(IFIX(ANO,AF,0.0
0005          RETURN
0006          END
```

111

3.71.- DESCRIPCION Y FORMATO DE LA FUNCION "VAL"

112

VAL (NPROP, SEXO)

Esta función obtiene el número de propiedad NPROP, para un sexo en una Entidad Federativa.

- Si NPROP es propiedad por posición la toma (FETCH)
- Si NPROP es propiedad explícita la busca en forma binaria en la zona de propiedades explícitas por posición ordenadas; si no, la busca en forma secuencial en la zona de propiedades explícitas desordenadas.
- Si NPROP es propiedad computada, la computa (entre ellas estan las propiedades codificadas, las funciones, fórmulas y tablas).

3.72.- LISTADO DE PROGRAMA DE LA FUNCION "VAL"

```
DOS FORTRAN IV V30N-FD-435 3-0          VAL              DATE  31/10/79      TIME  17.19.23      PAGE 0001

0001            INTEGER FUNCTION VALV2 INPROP,SEXO,PCSO
0002            IMPLICIT INTEGER*2 (I-Z)
0003            COMMON /BENEFC/VENTEE(2000
0004            COMMON /ALMREC/CIFRAS(3
0005            COMMON /PARAME/PARAMETRO
                C VENTEE ES EL VECTOR DONDE ESTA GUARDA TODO EL CUERPO DE LA ENTIDAD FEDERATIVA
                C PUS ES LA ENTRADA DE VENTEE DONDE EMPIEZA LA EDAD QUE NOS INTERESA.
                C SEXO  0- EL SEXO SEA MASCULINO DE LOS CUALES A QUE SUCEDO SON PROPS. IMPLI-
                C       CITOS,EXPLICITOS,CODIFICADOS,ETC.
                C                                                                    113
                C AQUI A LA PISO SON LAS CODIFICADAS DE CRAVIOTO.
0006            IF(VAPROP.GT.8)GOTO 10
                C VEO SI ES IMPLICITA O EXPLICITA.
0007            IF(VAP=CP.LT.CIFP-CRIE.OR.VAPROP.GT.CIFPRECISION*10+9)
0008            VALEVALC2VAPROP,SEXO,PCSO
0009            RETURN
                C AQUI LAS CODIFICADAS.
0010        10  IF(VAPROP.LE.8*CCCCC IC 20
                C VEO SI SON PRELLENAS.
0011            IF(VAPROP.LT.CIFPCTEE.OR.VAPROP.GT.CIFP*CIFRECISION*10+9)
                C ESTAS SON LAS TABULADAS.
                C FALTA AGREGR LAS TRANSFORMACIONES PARA LAS TABULADAS.
0012            VALCC
0013            RETURN
                C ESTAS SON LAS DE CRAVIOTO.
0014        20  APROPIVAPROP=3CCC
0015            VALEVALC2VAPROP1,SEXO,PCSO
0016            TRANSFC1VAPROP1-8CCCC/125+1
0017            IF(TRANSF.EC.SETRANSFA
0018            NTRANSF=TRANSF
0019            NTRA2=1
0020            IF(TRANSF.EC.
0021            VALECCCIFVAL,PARAPIAIO,PARAMIAZO,TRANSF)
0022            RETURN
0023        30  VALFE
0024            RETURN
0025            END
```

3.74.- LISTADO DE PROGRAMA DE LA FUNCION "VALO"

```
DOS FORTRAN IV 3.0JN-FD-429 5-8          VALO          DATE  31/10/79      TIME  17.15.51      PAGE 0001

0331            INTEGER FUNCTION VALO*2(NPROP,SEXO,COSO
                C ESTA RUTINA EXTRAE EL VALOR DE LA PROP NPROP,CORRESPONDIENTE A LA PERSONA
                C QUE EMPIEZA EN POS.
                C EL VALOR DE LA PROPIEDAD LO DA SEGUN SEXO, 1 MASCULINO ,2 FEMENINO Y 3
                C AMBOS
                C CUANDO ESTA FUNCION ES LLAMADA NPROP ES MAYOR O IGUAL QUE DIFPROS10 Y
                C MENOR O IGUAL QUE DIFPROS20, ES DECIR ES IMPLICITA O EXPLICITA.
0332            DIMENSION VENTEC(2),VENTECS(500
0333            COMMON /VALPRO/DIFPROS10
                C VEO SI ES IMPLICITA O EXPLICITA
0334            IF(NPROP.GT.DIFPROS10)GOTO 10
                C FUE IMPLICITA
                C VEO SI ESTA EN ESTA RUTINA EN PARTICULAR.
0335            IF(NPROP.LT.VENTEC(3)+50.OR.NPROP.LT.VENTECPOS+NMAGO) GO TO 201
0336            IF(POS-15+NPROP-VENTEC(2)+500*2
0337            GO TO 202
                C FUE EXPLICITA
                C VEO SI PUEDE SER EXPLICITA ORDENADA.
0338        10  IF(VENTECPOS+4)12
0339            IF(VENTECPOS+4
0340            IF(NPROP.LT.VENTEC(1)).OR.NPROP.GT.VENTECPOS+NMAGO) GO TO 30
                C PUEDE SER EXPLICITA ORDENADA Y BUSCO EN TABLA BINARIA
0341        12  IMPTE=3)C/2
0342            IF(IMP.EQ.CODE TO 21
0343            IMPTE=3+T)
0344            IF(NPROP-VENTEC)))*2)15,20,12
0345        15  IMPIP
0346            GO TO 12
0347        16  IMPIP
0348            GO TO 12
0349        20  IMPIP
0350            GO TO 100
0351        25  IF(NPROP.EQ.VENTEC(I))GO TO 100
0352            IF(NPROP.NE.VENTEC(I))GO TO 25
0353            IMPIP
0354            GO TO 100
                C SOLO FALTA BUSCARLA EN EXPLICITAS DESORDENADAS
0355        30  IF(VENTECPOS+1)
0356            IF(VENTECPOS+100
0357        35  IMPII+1
0358            IF(NPROP.EQ.VENTEC(I))GO TO 100
0359            IF(I-(VENTEC(2)*2))35,35
                C YA LOCALICE LA EXPLICITA. SI ME VIENE IMPAR CASTILLA ANTES DEL VALOR DE LOS
                C HOMBRES
                C EL SEXO ES UNO O DOSERES, POR LO QUE RESTO, PARA A AMBOS,
                C SI NO PIDIERON POR MUJERES O HOMBRES. SOLO CUENTA EL TOTAL REGISTRADO.
0360        100 IF(VENTEC(I+2),EQ.-1C.AND.SEXO.NE.3)GO TO 200
0361            IF(SEXO.EQ.CODE TO 110
0362            VALO=VENTEC(I+1)*2)C
0363            RETURN
0364        110 IF(VENTEC(I+2).NE.-1)GO TO 115
0365            VALO=VENTEC(I+1)
0366            RETURN
0367        115 VALO=VENTEC(I+1)*(VENTEC(I+2)
0368            RETURN
0369        200 VALO=0
0370            RETURN
0371            END
```

3.75.- DESCRIPCION Y FORMATO DE LA FUNCION "LLELIV"

LLELIV (LITEX, LL, PREVAL, i)

Llena la línea i del arreglo PREVAL a partir de la línea de texto LITEX. Este es un arreglo unidimensional que contiene caracteres, uno en cada byte.

Ejemplo:

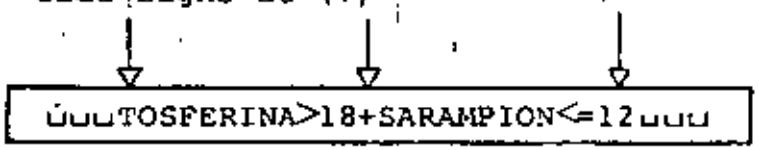
TOSFERINA > 18 + SARAMPION < 12

PREVAL. Arreglo de la pregunta a valores (a Entidades)

LL. Es el byte donde el texto acaba.

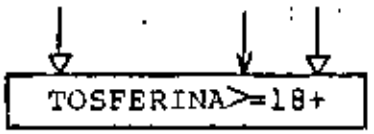
Forma en que trabaja: En la línea (i) del texto LITEX:

1).- Pone una ▽ en las puntas no blancas y en cada signo de (+)



2).- Para cada (j) texto entra un par de ▽

a) Busca las relaciones > = < * y les pone



Si no hay coma, seguir lo que dice d).

Si hay coma:

i) mete lo que se encuentre entre ↓ y ⇑ a PREVAL(i,4*j), previamente convertido a binario; y

ii) mete lo que se encuentre entre ⇑ y ↓ a PREVAL(i,5+(j-1)*4), previamente convertido a binario.

e) Aumenta el contador de ramas del OR

3.- Guarda el contador 2e) en PREVAL(i,1)

4.- Si la línea tiene un . , guarda un cero en PREVAL(i,1)

NOTA: Quien llame a LLELIV sabe si leyó un punto (ya acabó), porque guardó en este caso un cero en PREGU(i,1).

SECCION 4.- PROTECCION DE ARCHIVOS DEL BANCO DE DATOS.

Fundamentalmente se tendrán 2 clases de archivos.

4.1.- Archivos de Programas.

4.2.- Archivos de Datos.

4.1.- Archivos de Programas (Sistema Operativo del Banco).

4.1.1.- Los programas coexistirán con el S.O. del Software del compulador "in-line" para satisfacer permanentemente los requerimientos que se hagan del Banco. Radicarán en Disco y en Memoria Central.

4.1.2.- SOPORTE.- Son copias de los programas que radican en disco y memoria central y estas seran:

1.2.1.- Una copia de programas simbólicos en cinta que existirá en el Centro de Cómputo.

1.2.2.- Una copia de programas simbólicos en cinta que existirá en la cintoteca.

1.2.3.- Una copia de programs simbólicos que existirá en la bóbeda de seguridad.

4.1.3.- ACTUALIZACION.- En el transcurso del día o día los programas "in-line" podrán sufrir modificaciones, en cuyo caso, por la noche se obtendrá una copia en cinta de los programas conteniendo las modificaciones, dándose de baja la cinta con programas anteriores del Centro de Cómputo. Pasado un tiempo (una semana podría ser) si los cambios de los programas no reportaron fallas se

sacará un duplicado para la cintoteca y otro para la bóbeda conservándose un anterior y la última versión en cintoteca y la bóbeda.

4.1.4.- PROTECCION.- Los archivos tanto en cinta como en disco estarán protegidos por una etiqueta cuyo contenido es el nombre y/o clave del archivo que se guardará confidencialmente para que el acceso sea permitido solo por quienes conocen este nombre y/o clave. La fecha de caducidad se dará con fecha juliano 99/365 para que el sistema no de de baja estos archivos por expiración de fecha de caducidad. Esta protección se hará ya sea en el momento de carga a disco o de copia a cinta agregando el comando que se indica a continuación en la corrida que se haga para la carga o copia a cinta.

4.1.4.1.- Comando a agregar en la corrida de carga de cinta a disco:
//DLBLØ SYS, * (etiqueta), 99/365
etiqueta = 44 caracteres para nombre y/o código del archivo.
99/365 = fecha juliana equivalente al 31 de diciembre de 1999 calendario normal.

4.1.4.2.- Comando a agregar en la corrida de copia del archivo a cinta:
//TLBLØ SYSXXX, "etiqueta", 99/365
etiqueta = idem.
99/365 = idem.

4.2.- Archivos de datos (Banco de Datos).

4.2.1.- Los datos radicarán "in-line" en memoria de discos para satisfacer permanentemente los

requerimientos que se hagan del Banco.
Los programas del Banco manejarán los datos de los diferentes años como un solo archivo.

4.2.2.- SOPORTE.- Son copias en cintas de los archivos de datos del Banco que radican en memoria de disco. En cinta una copia del Banco de Datos se convertirá en un multiarchivo de cinta conteniendo el final de los datos de un año un EOF (fin de archivo) y al final de todos los años de datos un doble EOF.

2.2.1.- Una copia en cinta de este multiarchivo de datos se mantendrá en la cintoteca.

2.2.2.- Una copia fiel de la anterior se mantendrá en la bóbeda de seguridad.

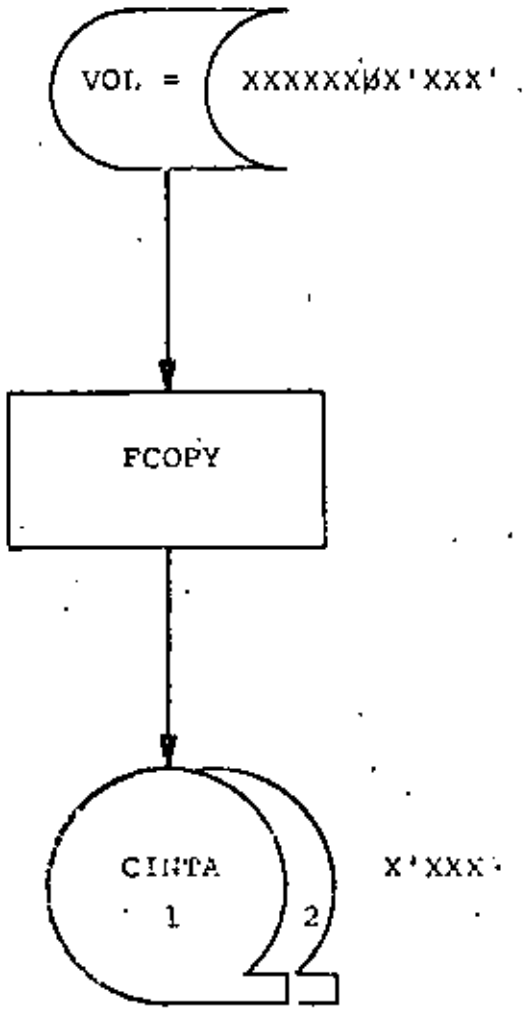
4.2.3.- ACTUALIZACION.- Como la actualización de estos archivos se hará con autorizaciones del responsable del Banco de Datos, los datos que se agregen o se quiten llevan la garantía de haberse revisado previamente, por tal motivo cada vez que se actualice el Banco se sacará una copia para la cintoteca y otra para la bóbeda de seguridad; es de recomendarse se conserven la última y 2 anteriores dándose de baja la cuarta mas antigua siempre.

4.2.4.- PROTECCION.- La misma indicada en 4.1.4. para los archivos de programas.

4.3.- Rutinas para producir los soportes y carga de archivos.

4.3.1.- <u>DISK-TO-TAPE.</u>

```
//  JOBbFCOPY
//  UPSIb10000000
//  ASSGNbSYS004,X'280', ALT
//  ASSGNbSYS005,X'155'
//  EXECbFCOPY, SIZE =  AUTO
//  UTD
/   *
/   &
```



A P E N D I C E "A"

TABLA DE CODIGOS DE PROPIEDADES

IDENTIFICACION	NUMERO DE PROPIEDAD	C O N C E P T O
00		ENTIDADES FEDERATIVAS
	00001	AGUASCALIENTES
	00002	BAJA CALIFORNIA NORTE
	00003	BAJA CALIFORNIA SUR
	00004	CAMPECHE
	00005	COAHUILA
	00006	COLIMA
	00007	CHIAPAS
	00008	CHIHUAHUA
	00009	DISTRITO FEDERAL
	00010	DURANGO
	00011	GUANAJUATO
	00012	GUERRERO
	00013	HIDALGO
	00014	JALISCO
	00015	MEXICO
	00016	MICHOACAN
	00017	MORELOS
	00018	NAYARIT
	00019	NUEVO LEON
	00020	OAXACA
	00021	PUEBLA
	00022	QUERETARO
	00023	QUINTANA ROO
	00024	SAN LUIS POTOSI
	00025	SINALOA
	00026	SONORA
	00027	TABASCO
	00028	TAMAULIPAS

IDENTIFICACION	NUMERO DE PROPIEDAD	C O N C E P T O
	00029	TLAXCALA
	00030	VERACRUZ
	00031	YUCATAN
	00032	ZACATECAS
	00033	ESTADOS UNIDOS MEXICANOS
01	SEXO	
	00041	MASCULINO
	00042	FEMENINO
	00043	AMBOS
02	EDAD	
	00051	MENORES DE UN AÑO
	00052	0-1 AÑOS
	00053	0-2 AÑOS
	00054	0-3 AÑOS
	00055	0-4 AÑOS
	00056	0-5 AÑOS
	00057	0-6 AÑOS
	00058	0-7 AÑOS
	00059	0-8 AÑOS
	00060	0-9 AÑOS
	00061	0-10 AÑOS
	00062	0-11 AÑOS
	00063	0-12 AÑOS
	00064	0-13 AÑOS
	00065	0-14 AÑOS
	00066	0-4 AÑOS
	00067	5-9 AÑOS
	00068	10-14 AÑOS
	00069	5-14 AÑOS
	00070	1-14 AÑOS
	00071	12-14 AÑOS
	00072	12 AÑOS
	00073	13 AÑOS

IDENTIFICACION	NUMERO DE PROPIEDAD	C O N C E P T O
	00074	14 AÑOS
	00075	1 AÑO
	00076	2 AÑOS
	00077	3 AÑOS
	00078	4 AÑOS
	00079	5 AÑOS
	00080	6 AÑOS
	00081	0-29 DIAS
	00082	1-11 MESES
	00083	6-9 AÑOS
	00084	4-5 AÑOS
	00085	6-14 AÑOS
	00086	MENORES DE 7 AÑOS
	00087	7 AÑOS
	00088	8 AÑOS
	00089	9 AÑOS
	00090	10 AÑOS
	00091	11 AÑOS
	00099	EDAD NO ESPECIFICADA
03	TALLA	
04	PESO	
	02499	POBLACION
05		CARACTERISTICAS SOMATICAS. NATALIDAD
	02540	NUMERO DE NACIMIENTOS
06		CARACTERISTICAS SOMATICAS MORTALIDAD
	02549	MORTALIDAD INFANTIL
	02550	FETAL (NACIDOS MUERTOS)
	02551	NEONATAL (MENOS DE 28 DIAS DE EDAD)
	02552	POSTNEONATAL (DE 28 DIAS A 11 MESES DE EDAD)
	02553	INFANTIL MENORES DE UN AÑO

ARQUITECTURA BASICA DEL SISTEMA DE COMPUTO DONDE RESIDE EL BANCO DE DATOS EN SU FASE INICIAL DE OPERACION.

- IBM 370/135 con 512 KB de memoria real.
- 6 unidades de disco de 300 MB c/u, controlador de disco 3333.
- 2 impresoras lineales con capacidad de impresión de 1200 L.P.M. modelo 1403.
- 3 Unidades de cinta modelo 3410 1 controlador de unidades de cinta modelo 3411.
- 1 controlador de terminales modelo 3704 6 terminales con pantallas de CRT.
- 1 lectura de tarjetas modelo 3505.
- 1 consola maestra modelo 3215.

CASOS PRACTICOS DEL DISEÑO DE SISTEMAS DE INFORMACION

TECHNOLOGY AND ARCHITECTURE FOR DATA MANAGEMENT: AN OVERVIEW AND PERSPECTIVE

(revista COMPUTER, MAYO 1979)

AGOSTO, 1980

COMPUTER

May 1979 Volume 12 Number 5 (ISSN 0018-9162)

THEME FEATURES

COMPUTER

Cover design: Franz Yanni

8 **Guest Editor's Introduction: Technology and Architecture for Data Base Management—An Overview and Perspective** *Harut Barsamian*
Rapid growth of data bases in end-user applications and as powerful software development tools will make data management systems one of the greatest challenges of the 1980's and a landmark of that decade.

12 **Storage Technology: Capabilities and Limitations** *A. S. Hoagland*
The disk remains the primary form of mass storage. Despite competing technologies, improvements in disk technology will ensure its widespread use throughout the remainder of the century.

19 **Analysis of Memory Hierarchies for Sequential Data Access** *Terry A. Welch*
Analytic modeling techniques permit system designers to compare memory-system alternatives quickly and easily. An example is the use of buffers—or caches—on disks.

27 **Current Trends in Data Base Systems** *O. A. Champine*
Tomorrow's data base systems will be organized around a common architecture, data distribution techniques, and new specialized processors.

42 **System R: A Relational Data Base Management System** *The System R Group*
A relational approach makes this experimental data base management system unusually easy to install and use. Some of the decisions made in System R design in order to enhance usability also offer major bonuses in other areas.

SPECIAL FEATURES

50 **A Keynote Address on Concurrent Programming** *Per Brinch Hansen*
Delivered at COMPSAC 78, this address draws a parallel between the major development phases of the first 20 years of concurrent programming and the present challenge of distributed computing.

58 **Microprocessors in Japan—Status in 1978** *Ryoichi Mori, Morihiko Tajima, Yoshikuni Okada, and Hiroaki Tajima*
By the third quarter of 1978, ten Japanese companies were producing some 80 types of microprocessors, including 61 original products. Both the rate of new product introductions and the directions of these developments are impressive.

64 **Microcomputer Applications in Japan** *Ryoichi Mori, Yuzo Kita, Iwao Morishita, Akio Taja, Akio Kokubu, Yoshikuni Okada, Shunichi Uchida, Kiyoto Ohkawa, and Seishi Nishikawa*
JEIDA, the Japan Electronic Industries Development Association, recently surveyed the state of the applications art in Japan. The most striking features of the study were its scope and the richness of the answers obtained.

76 **Conference Report: VLSI Architecture, Design, and Fabrication** *Philip M. Neches*

79 **Open Channel: You Can't Just Plug Your Computer into the Wall!** *James V. Dinsey*

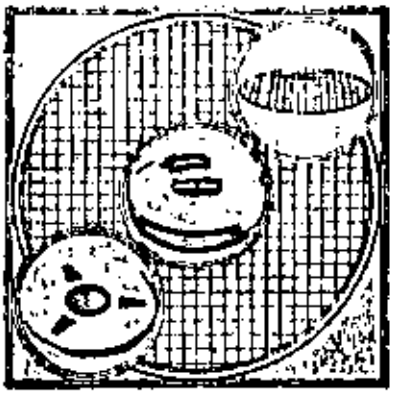
DEPARTMENTS

4 Special Messages
82 New Products
86 New Literature
87 IC Announcements
88 Microsystems Announcements
89 New Applications
91 Update
95 Classified Ads
99 Call for Papers
100 Calendar
102 Book Review: *The Architecture of Concurrent Programs*
104 Advertisers/Product Index
107 The Bookshelf

Coming Next Month: Circuit Switching

Reader Service Cards and Order Forms, pp. 105-106.

Rapid growth of data bases in end-user applications and as powerful software development tools will make data management systems one of the greatest challenges of the 1980's and a landmark of that decade.



Technology and Architecture for Data Management: An Overview and Perspective

Guest Editor's Introduction

Harut Barsamian
Sperry Univac

The technology of electronic computing is nearly 40 years young. Industry analysts tend to describe computer evolution in terms of "generations," each characterized by the predominant hardware technology used for logic—tubes, transistors, or ICs. Such characterization is superficial and reflects a narrow point of view. It does not depict the more substantive technological breakthroughs that have contributed to the spectacular growth of the computer industry.

The 1940's were the pioneering years; scientific foundations and original architectural concepts were established. Although the discoveries and achievements of any one period have been advanced and expanded in subsequent years, each past decade (not necessarily at discrete calendar boundaries) has added a new dimension to the genealogy of the computer. During the 1950's the engineering and manufacturing feasibility of computers was proven and their commercial viability was established. The era of the computer "family" and contemporary operating systems began in the 1960's. The 70's became the decade of computer communications, signified by the rapid expansion of remote computing, distributed processing, and computer networks. And the 1980's? An extrapolation of computing's evolutionary path, based on today's realities—the rapid growth of data bases in end-user applications and as powerful tools for software development—indicates that the development of efficient data management systems will be one of the 80's greatest challenges as well as a landmark of that decade.

Data management systems are quite complex in nature, and problem solutions tend to cross the boundaries of various disciplines. Technology, storage hierarchies, architecture, and data definition and manipulation languages are intimately related to the design and performance of data management systems. The need for further exploration of this interdisciplinary synergism set the topical framework for the 17th Lake Arrowhead Workshop on Data Management and Storage Hierarchies, held in September 1978 and sponsored by the Western Area Committee of the IEEE Computer Society. The authors contributing to this special issue were key participants in that workshop.

Storage technology. A data management system can be viewed as a multilayered structure, with storage devices comprising the nucleus (top, facing page). Compared to most technologies in the computer industry, contemporary storage devices have the highest degree of technological maturity, with well-understood design criteria and trade-offs and progressively improving cost/performance indices. Storage technology, which provides the physical host environment for data bases, remains the main factor in determining the cost and performance of data management systems.

Magnetic disks, more than any other storage technology, have dominated the on-line mass storage area for nearly two decades. They will continue to do so in the foreseeable future, perhaps thoughout the rest of this century. None of the existing or proposed storage technologies—optical and electron beam memories, magnetic bubbles, or charge coupled devices—currently exhibit a totality of characteristics superior to disks. A. S. Hoagland's "Storage Technology: Capabilities and Limitations" offers an insightful analysis of the technical and economic aspects of storage technology.

0018-9162/79/0500-0008$00.75 © 1979 IEEE

The most often stated drawback of rotating magnetic disk devices is access time. The difference in access time between disks and main memories (presently in the range of four to five orders of magnitude) is the well-known "access gap." The instinctive human fear of "gaps" seems to trigger a constant search for technologies that can fill this "gap." But is there a cost-effective technology available today to fill it? For a time magnetic drums and head-per-track disks were considered the only viable "gap fillers," although they were unable to gain a permanent position in storage hierarchies across a wide spectrum of applications. In recent years magnetic bubbles and CCDs have become the most often cited "gap fillers." However, the cost and performance of these technologies, compared to disks and RAMs, have not yet reached a level making them unanimous choices as "gap fillers" across the spectrum of computer system applications. In spite of some optimistic claims, these technologies may be viable for only a narrow band of applications. Such limited use of CCDs and bubbles does not make them indisputable and true "gap fillers."

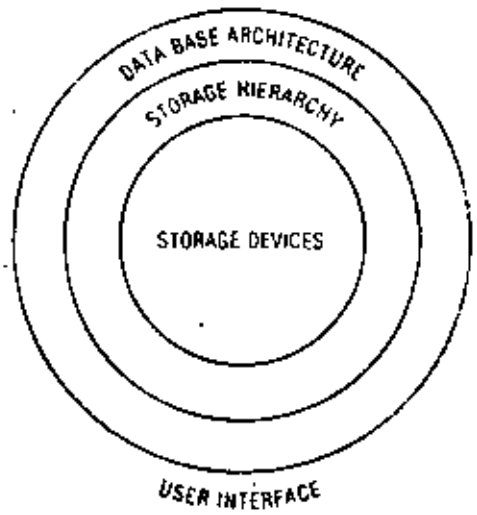
Both disks and RAMs have become fast-moving targets with proven records. The technology which might become the true "gap filler" will have to offer the best combination of characteristics, i.e., CCD performance, the cost/bit potential of magnetic bubbles, and (if possible) non-volatility. By all indications to date, the computer industry must wait a while for this messiah to arrive...

Is there a viable "gap filler" technology today?

Storage hierarchies. Efficient organization and management of large storage spaces are becoming the focal issue in data management system design. In general, large storage spaces are organized in a hierarchical fashion, with the various levels implemented in different technologies for mainly economic reasons. Storage hierarchies hosting data bases exist in a three-dimensional space defined by access time, cost, and capacity. The successful storage hierarchy design achieves an optimum balance between access time and cost, while the values of these dimensions always maintain a mutually inverse relationship. The designer seeks the fastest possible access time at the lowest possible cost/bit for the entire storage hierarchy. The third dimension, total data base storage capacity, is primarily determined by start-up costs and end-user application requirements.

The performance of storage hierarchies depends on the storage access patterns and program behavior in actual run-time environments.

Thus, the storage hierarchy design process must involve the making of optimum tradeoffs not only in device selection, but also in space allocation and data replacement. The prevailing methodology for investigating program behavior is computer simulation, often supplemented by data measured on "representative" programs, i.e., benchmarks. However, these techniques are sometimes impractical and always complex and costly, and so remain out of reach for the majority of the design community. As a result, most design choices and trade-offs are generally based on intuitive assumptions and empirical approximations using sparse, inconclusive statistical data.



Mathematical modeling techniques can provide quick results and are an alternative to lengthy and costly simulation runs. Using algebra, calculus, and basic physics, relatively simple models of storage devices and information flow within the hierarchy can be constructed. Such models allow exhaustive yet rapid investigation of storage hierarchies, including optimization of critical parameters. The accuracy of the results obtained through mathematical modeling is relatively low compared to simulation, but such results are useful for establishing first-level correlation between critical parameters and for pointing out the direction further research should take. "Analysis of Memory Hierarchies for Sequential Data Access," by Terry Welch, is a vivid demonstration of the practicality and merits of mathematical modeling as a tool for trade-off analysis in the design of storage hierarchies.

Data base architecture. The proliferation of data bases, specifically in on-line and interactive transaction processing applications, is increasing the demand for high performance, system availability, and "easy to use" user interfaces. Moreover, the trend toward distributed systems will inevitably lead to the distribution of data bases, thus adding new complexities to the perennially critical design problems of data integrity, recovery, security, and privacy. Indeed, monumental challenges and a myriad of non-trivial tasks will confront data management system architects in the years ahead, and perhaps throughout the rest of this century. George Cham-

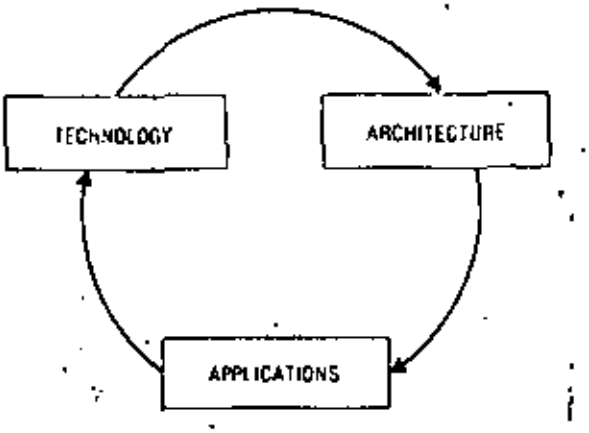
pine's "Current Trends in Data Base Systems" presents an in-depth overview of the present status of data base systems, and outlines a framework for future solutions of the fundamental problems in data base system architecture.

The architectures of various data base systems are distinguished mainly by the data models they are designed to support. Various data base models—network, hierarchical, relational—require a varying degree of user involvement for access, formatting, and control of underlying storage hierarchies. The network data model based on the Codasyl specification is the most widely used in present data base systems. The relational data model, however, continues to gain attention since it appears to be more user-oriented. "System R: A Relational Data Base Management System," by M. M. Astrahan et al., describes the main features of a relational data base management system. Although considered an experimental project, System R suggests several sound architectural ideas that might be refined and transported into the commercial domain of future data base systems.

Advances in LSI and storage technologies will provide sound economic justification for specialized data base machines.

Relatively low performance is the main drawback of relational data bases, since the system must perform exhaustive search operations to satisfy queries. In general, however, low performance is inherent in any data base system because the architectural inadequacies of conventional general-purpose computers make it impossible to perform search operations efficiently. Although these inadequacies, manifested primarily in non-numeric operations, were recognized a long time ago, data base applications increasingly accentuate their criticality. Recent research efforts have achieved visible progress in the search for novel solutions—at both technological and architectural levels—to the problem of data management system efficiency (see the March 1979 special issue of *Computer*—"Data Base Machines"). Associative processing techniques may increase the performance of search and sort operations—key time-consuming functions in data base management systems. Dedicated and specialized processors show promising results in handling data access and storage hierarchy management tasks, thereby off-loading the central processor and increasing system throughput. Advances in LSI and storage technologies and the increasing cost of programming, trends bound to continue in the years ahead, provide a sound economic justification for specialized data base machines.

Like the growth of the information processing industry in general, we can view the evolution of data management systems as the forward motion of a



positive feedback loop, with economics remaining the driving force. Applications will expand continuously and stimulate new technological breakthroughs while architecture, although evolving more gradually, will continue to transfer the benefits of new technologies to the end-user. ■

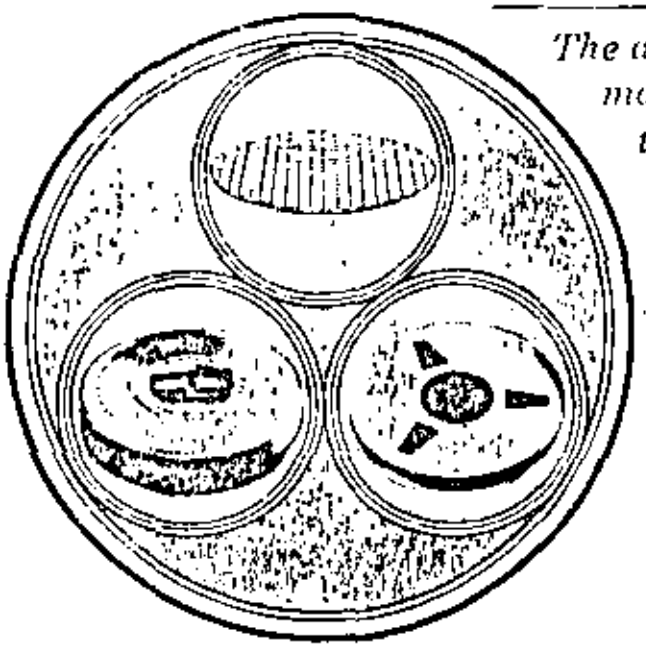
Acknowledgment

I would like to thank all the authors contributing to this special issue for their patience with me, and the reviewers of the papers, Lowell Amdahl, David K. Hsiao, Dan Marro, and Richard Mattson, for their time and effort. Thanks are also due to Portia Isaacson, technical editor of *Computer*, and Ted Lewis, associate technical editor, for their guidance and support.

I would also like to express special gratitude and appreciation to my secretary, Suzanne Anderson, whose unselfish support and assistance made this long crusade—the Lake Arrowhead Workshop, the follow-up session at COMPCON 79 Spring, and the completion of this issue—possible and worthwhile.

Harut Barsamian is manager of Advanced Systems Planning for Minicomputer Operations at Sperry Univac, Irvine, California. Previously with NCR, he was the head of the Advanced Studies Section in the Data Processing Division, and later Senior Consultant at the NCR/GDC Advanced Systems Laboratory. Prior to NCR he was with Raytheon Company and was a consultant to the Computer Science Department of the Rand Corporation.

Barsamian received his Engineer Electric degree (with honors) from the Polytechnic Institute of Yerevan, Armenia, USSR. He subsequently completed studies for a doctoral-level degree in computer sciences at the Institute of Precise Mechanics and Computer Engineering of the USSR Academy of Sciences, Moscow. The author of numerous papers and the holder of two US patents, Barsamian is a member of ACM and IEEE. He has organized and chaired technical sessions at several IEEE and ACM annual conferences and National Computer Conferences. He was an IEEE Distinguished Visitor in 1972-73 and 1973-74, and an ACM National Lecturer in 1976-77.



The disk remains the primary form of mass storage. Despite competing technologies, improvements in disk technology will ensure its widespread use throughout the remainder of the century.

Storage Technology: Capabilities and Limitations

A. S. Hoagland
IBM San Jose

Improving the performance and extending the utility of information processing requires access to ever larger volumes of data. Thus, the impact of progress in storage technology on the overall growth of the information processing industry is immense. By advances that continually reduce the cost of storage it becomes feasible to develop and place more and more applications on computer systems by permitting both a larger system data base and more sophisticated control and applications programs. Unfortunately, the term "mass storage" has no precise technical meaning, being all too frequently used to describe on-line peripheral storage of a size bigger than a user currently believes he can justify. Once the user's system is upgraded to that level, however, as technical advances over time allow capacity to be increased for a fixed cost, this amount of storage is barely considered adequate, much less "mass." In discussing hardware, the term is most commonly associated with mechanical storage devices. While solid-state memory continually provides more and more bits per chip, the rate at which storage capacity can be absorbed grows even faster, and thus it seems electronic memories will never eliminate the use of auxiliary storage devices. As storage costs decrease, applications expand, the market grows, and product volume increases; the resulting cost reductions in turn reinforce the cycle. In addition, it then becomes feasible to increase investment in technology, which results in a further improvement in cost/performance. It is thus important to understand storage technology trends if one is to anticipate the future.

Overview

With the increased circuit densities now being realized, the CPU is shrinking in size, and this, coupled with the insatiable demand for more and more storage, now begins to invoke a visual impression of a "computer center" as only an assembly of disk and tape drives and associated media storage libraries; "peripheral" may now be a more appropriate adjective for the CPU than for storage. (Let the "chips" fall where they may.) Figures 1 and 2 give some illustration of the disk and tape world today.

The use of storage technologies depends on three principal factors:

- cost per bit,
- access time, and
- entry cost (capital required).

Reduced cost per bit in all technologies derives primarily from an increase in density on the material being used for storage. In each technology, lower cost per bit is also associated with an increase in the physical size of the basic storage module, the capacity-cost breakpoints being a function of the particular technology. Thus, in low-end systems, where a fixed system cost constraint exists, product entry cost, more than capacity (or cost per bit), becomes the name of the game.

When storage technologies are plotted on a graph of cost per bit versus access time (Figure 3), a wide separation both in access time and cost per bit

0018-9162/78/0500-0012$00.75 © 1978 IEEE

becomes evident between electronic semiconductor memories and devices that rely on motion for data access. On the spectrum of access time, semiconductor memory is on one side and magnetic recording storage on the other, with numerous alternative technologies falling somewhere in the access-time gap between them. This gap is the carrot that has stimulated storage device developments based on electron beams, magnetic bubbles, optics, and so on. The access-time gap, reflecting the difference between electronic and mechanical times, may be something of a trap—perhaps "crevasse" would be a better term than "gap," because it carries the implication of a hazard—since existing technologies are in reality undergoing continual cost and performance improvements. Given such a broad access-time range, large differences in speed would have to be accommodated in a system design in any event.

It is frequently said that there are mature technologies and new technologies, with the inference that the rate of progress will be much greater in a new alternative. This can be misleading. For any technology to succeed, it must offer something superior to a current technology at the time of its actual introduction. And existing technologies are moving targets. Thus one cannot begin at a rudimentary state with a new approach, but must develop the technology at least far enough to gain initial market acceptance.

In the early days of computers, some rust (iron oxide to be specific) on a tape or disk and a small "horseshoe" with a coil to serve as a magnetic head could have almost sufficed, since at that time no other storage option (except for punched cards) really existed. On the other hand, in the long exploration of optical memory approaches (where the basic knowledge predates magnetic recording), one designed to wavelength limits right from the beginning, the problem being rather the lack of a suitable read/write material. We now see this same degree of sophistication in magnetic bubble work, where there is a large scientific effort. Only a major advance over competing capabilities can have meaning, since the "ball game" has now been underway for some time. While no technology can be economically improved indefinitely, technological potential is not so much a function of newness as it is of intrinsic physical and engineering limits.

The storage hierarchy

Since one must trade off improved access time against cost per bit, a memory/storage system generally consists of a hierarchy—a tradeoff mix of device performance capabilities.

The relatively high cost of electronics in years past made us focus on architectures to keep the CPU busy. The term "access gap" came into vogue to highlight the mismatch between the main memory's speed and the access time of mass storage. This access-time differential is five or more orders of

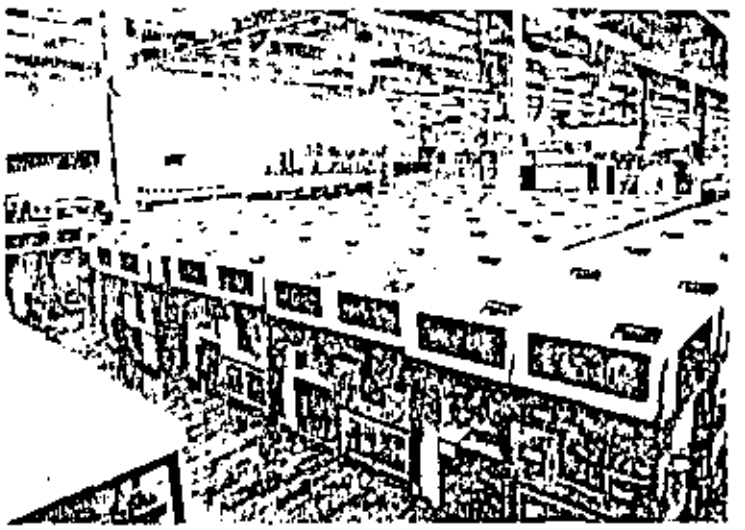


Figure 1. Disk drives ready for shipment. Demand is obviously high.

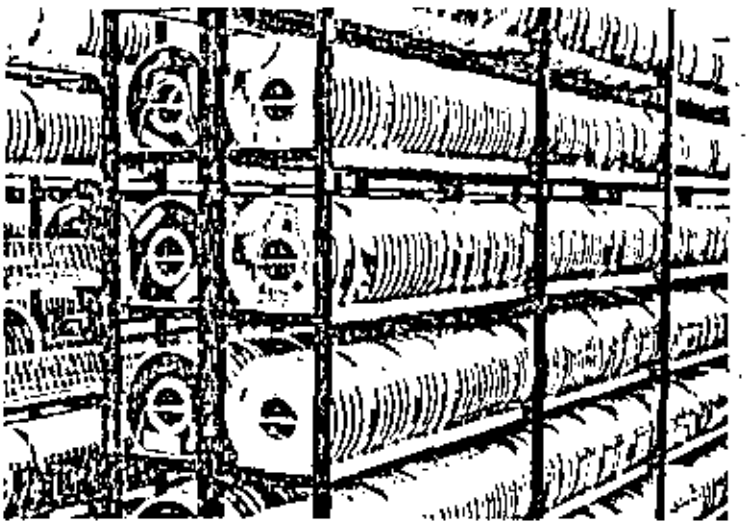


Figure 2. A typical tape library of today.

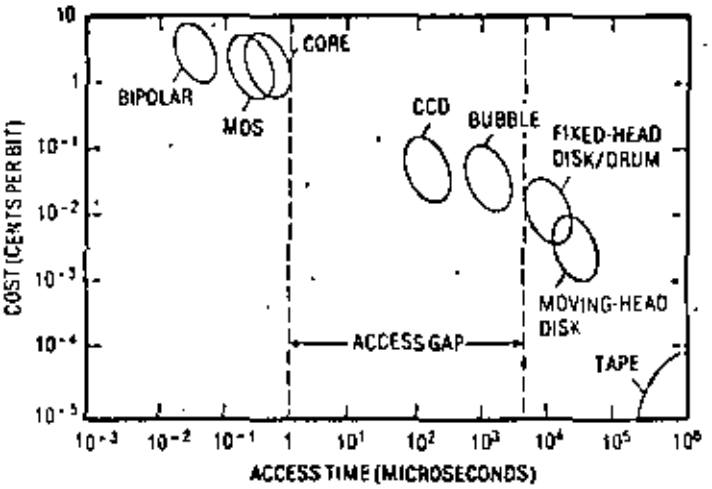


Figure 3. Storage technologies—cost per bit versus access time.

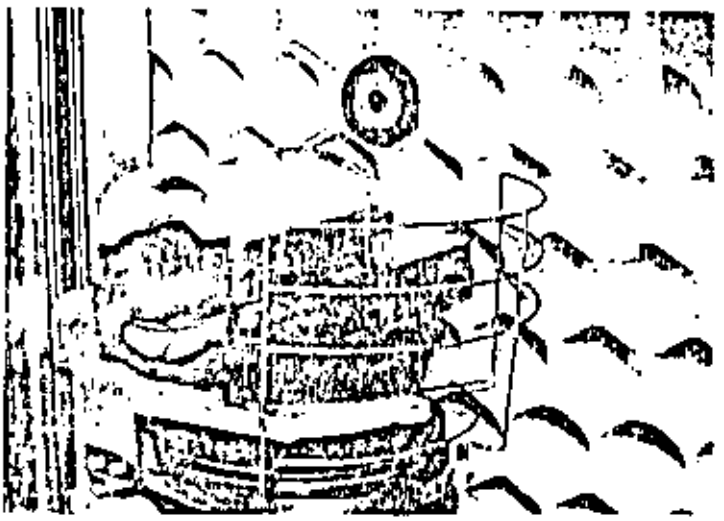


Figure 4. A 3850 cartridge library.

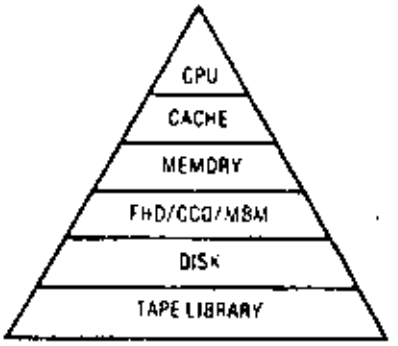


Figure 5. The classical storage pyramid.

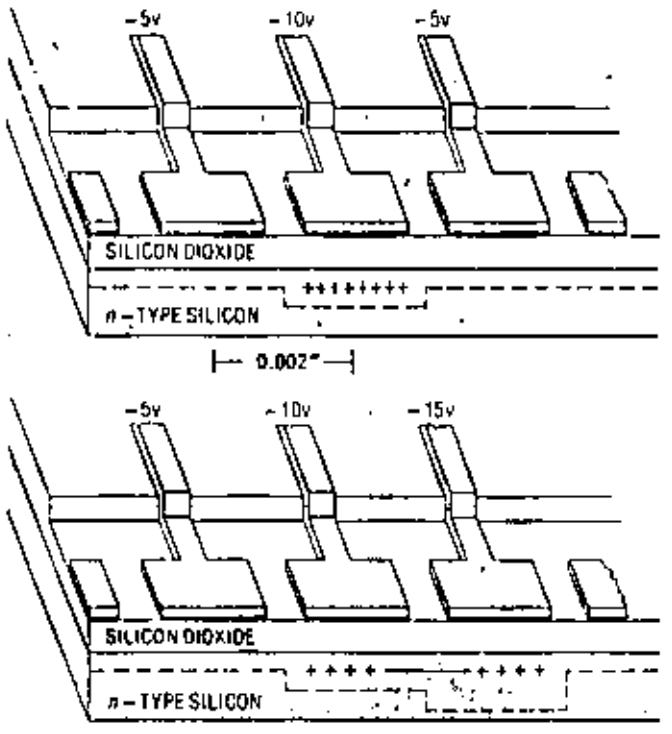


Figure 6. CCD structure.

magnitude and has called for special paging techniques. However, with continual reductions in the cost of a CPU, the criterion that it must be kept busy to be economic does not now loom nearly as important. Rather, the optimized use of the data base store becomes the key factor.

The IBM 3850 (see Figure 4) has a staging scheme by which all data on the tape cartridges in the automated library are transferred via disk. An intermediate buffer (as with a fixed-head drum or as in the Memorex 3770, which uses CCD for this purpose) can make sense in transferring data from disk to main memory. However the amount of storage involved in this buffering will be a small fraction of that on disk. This is not the gap-filler market that has been the holy grail of technologists.

The classical storage pyramid is shown in Figure 5, indicating the quantitative allocation of data storage in terms of technologies. This diagram reflects the cost-per-bit access-time tradeoff. The actual hierarchy of storage capabilities is very sensitive to the applications being implemented. For interactive applications such as airline reservations, all the data must be available in a few seconds. Payroll processing can be scheduled, and in this type of job the required information can be staged up from slower-and-lower-cost storage devices in advance of its need. In certain real-time applications it is essential that the data be available effectively instantly. The management of the data flow up and down the hierarchy carries software burdens that can reduce the potential utility of additional levels. To date, this topic has been effectively treated and understood only by the simulation of actual job streams.

The system design challenge is formidable, and this article only enumerates the technology capabilities with which it must proceed.

CCDs and MBMs

The CCD is a memory design based on well established silicon technology. Being a shift-register derivative of RAM, the CCD obtains a cost advantage by storing information in packets of charge rather than by means of bistable circuitry (see Figure 6). The technological similarity, while a great asset, may in fact retard the wide acceptance of CCDs for storage because, compared to RAMs, the obtainable cost reduction of a factor of two to four may not be enough to offset the 1000-to-1 decrease in access time, except in very specific applications.

The MBM—magnetic bubble memory—also uses a shift-register organization. A major-minor loop arrangement is most common, where a word is transferred in bit-parallel fashion from or to minor-loop registers via a major loop through which the information is serially shifted into or out of the memory (Figure 7). Magnetic bubbles are small cylindrical magnetic domains formed in certain thin

magnetic films when an external magnetic bias field is applied normal to the plane of the film. These "bubble" domains are magnetized in the reverse sense to that of the film itself. Information is coded in terms of the presence or absence of bubbles. The bubbles are controlled by creating a magnetic field gradient through a rotating field operating in conjunction with permalloy artwork defining the bit cell locations. MBM is nonvolatile in contrast to CCD.

CCDs and MBMs depend on similar fabrication processes. Lithography will be a prime limiting factor for both, and they should experience similar learning curves.

The 256K chip seems indicated as the first point of market stabilization. For CCDs this density requires two-micron geometries, and for MBMs two-to three-micron bubbles. (The artwork for CCDs follows different rules than that for bubbles.) With unit capacities of this size, either CCDs or MBMs will provide a lower cost per bit than magnetic recording for small increments of memory.

A quick look at Figure 3 suggests that, if these projections hold, CCDs should be more attractive than MBMs where nonvolatility is not important—as, for example, in providing a front-end buffer store for disk—since they offer obvious access-time advantages. Where paging time is not critical, it would even appear premature to predict the early demise of the fixed-head disk. From the standpoint of cost per bit, neither CCDs nor MBMs appear a threat to moving-head disks.

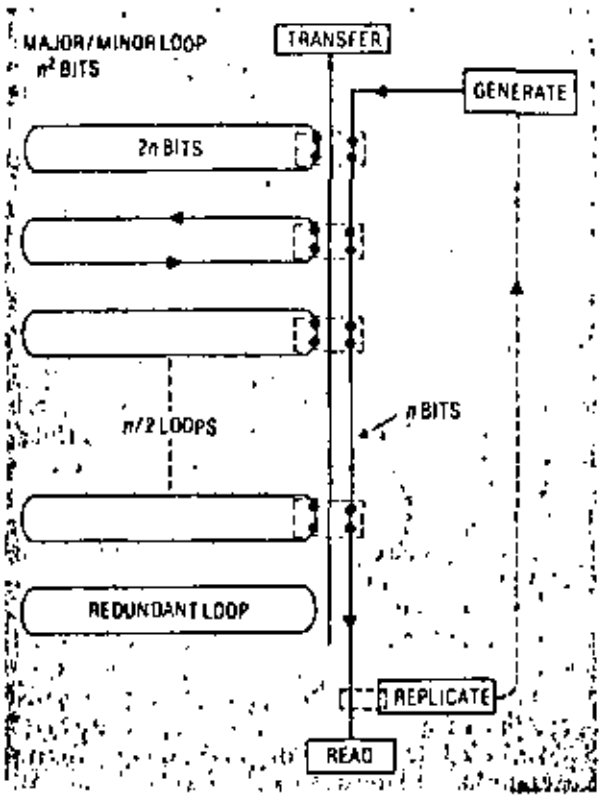


Figure 7. Magnetic bubble memory organization.

MBMs should fare better against the floppy disk in small systems, where total cost rather than cost per bit (i.e., capacity) is the key factor. Disk interchangeability is extensively used as a convenient cost-saving measure, and this unique feature complicates direct comparisons, since the floppy disk has a dual role, being used both for direct access storage and for loading and unloading programs and data. Another factor that may bear on small systems in the future is the rigid disk, since it more readily lends itself to high performance than the floppy. If suitable cost/performance tradeoffs can evolve from this higher level of technology, the rigid disk could represent an additional market factor.

Both CCD technology and magnetic recording intrinsically can offer multilevel storage, although such future potentialities are not seen to be a factor in current technical strategies. Based on historic technology trends, it appears that moving-head disk technology will maintain its relative cost advantage over CCDs and MBMs at least through 1985.

Among all the new storage technologies, magnetic bubbles are most likely to become established, mainly because they offer nonvolatility in small-capacity modules (and hence low entry cost)—not because they threaten and will displace the current technology mainstreams.

Optical video disk

Optical video disk storage, while not a read/write type of device (i.e., data cannot be updated in place), is attractive because it provides a higher recording density than magnetic recording and because, for read-only applications, the stored information lends itself to replication. In this sense it looks well suited to certain archival storage applications.

With this technology, a bright beam of modulated laser light is focused onto a spinning disk (either to produce a master from which copies are later reproduced or for directly recording upon an on-line storage medium). The recorded information can be retrieved later by illuminating the disk (or a copy) with a somewhat less intense beam and then detecting the modulation of light either reflected back or transmitted by the disk. A track pitch of about two microns is used in video disks, giving approximately 15,000 tracks per inch, and 15-mHz-bandwidth systems (minimum wavelength approximately one micron) have been developed.

Although the techniques used are optical, the design challenges are mainly mechanical and require the use of sophisticated tracking and focusing servo systems. Recording and playback take place through a protective overcoat layer, so dirt clinging to the disk exterior is always kept out of focus.

The recording material is a key factor when a postable (write-once, repeated-read) medium is desired for data storage. Verification during writing and rapid correction of recorded information require a recording material that needs no processing before

playback and that is impervious to cumulative degradation when subjected to continuous playback.

The ablation of thin metallic films has been the most extensively investigated technique for optical recording. In thermal recording, such as the ablation of pits in a thin layer of metal, the exposure threshold is well defined: it is either the melting or vaporization temperature. While a transparent overcoat will prevent signal degradation due to dust by keeping particles out of the focal plane, defects can also occur in the evaporated coatings that form the recording film. At the high densities being projected (approximately one thousand million bits per square inch) it is imperative that these coatings be very nearly pinhole free, since even a one micron imperfection will result in a significant loss of information. (Substrate defects can be masked by first covering the substrate with a thin organic layer.)

The potential of optical recording would be vastly improved if a reusable material could be found and if solid-state lasers could be used for recording and readback. This is not a new situation: the lack of a suitable read/write material has plagued optical data recording since it was first investigated.

Magnetic recording

Magnetic recording dominates mass storage technology. Why? Because it is a "mature" technology that refuses to age gracefully.

A user specifying a set of desirable storage attributes would demand:

- lowest cost per bit—much less than projections for CCDs or bubbles,
- a competitive marketplace based on numerous suppliers and a large choice of product offerings,
- capacities up to gigabytes,
- shelf storage
- many capacity/access-time options,
- read/write and nonvolatility, and
- broad environmental tolerances.

A manufacturer could not help but be impressed by the following engineering claims for a storage technology:

- relatively modest entry cost,
- multi-billion-dollar industry,
- established production processes,
- medium replaceable if desired—providing an additional market for media, and
- increases in demand for capacity occurring faster than storage density through technological progress—providing a growing market.

The research and development community is bound to be interested in a technology characterized by:

- density still far from ultimate limits (density has doubled approximately every two years and is projected to continue to do so through the mid-1980's),
- a wide choice of materials,
- fairly well-understood phenomena,
- completely reversible, inherently stable processes, and
- a wide range of engineering design choices.

Table 1.
Twenty years of progress in disk storage.

YEAR	DEVICE	CAP. (MB)	BITS/ SQ. IN.	4K BLK ACCESS (MS)	DOLLARS PER MB
1956	IBM 350	5	2K	925	$153
1964	IBM 2311	7.3	110K	113	75
1965	IBM 2314	29.1	220K	85	23
1970	IBM 3330-1	100	775K	43.3	8.30
1973	IBM 3330-11	200	1495K	43.3	4.85
1973	IBM 3340	70	1650K	39.6	7.50
1975	IBM 3350	317	3058K	39.6	2.25
		Newly	Announced		
1978	STC 8650	635	6100K	*	*
1979	IBM 3370	571	7500K	*	*

*Not yet shipped

What technology could fit all the above? Magnetic recording, of course. And it is exactly these perspectives that explain why magnetic recording of digital information is preeminent.

Magnetic recording of digital information has been utilized for 30 years and, in fact, was serving the storage needs of computers when the CPU was still based on devices called vacuum tubes. Magnetic recording is not only the most enduring but also the most pervasive of digital storage technologies. As seen in Table 1, disk storage density has increased from 2000 bits per square inch (Ramac-1956) to 7.5 million (IBM 3370)—more than three orders of magnitude in 20 years—and density increases should continue at this rate in the 1980's. Magnetic recording activity is worldwide in scope and derives tremendous momentum from the role it plays in such applications as video, audio, and instrumentation recording, in addition to the computer storage field.

One might consider the disk, in terms of solid-state terminology, as a wafer 14 inches in diameter (equivalent to 10,000 chips) that is processed and used as an entity (therefore, perfect "wafers" are the rule). Bit cell length is currently about a micron, with a head-medium "dimension" of .1 micron. Figure 8 shows how recording geometry has scaled down with increasing linear density.

With all these advantages going for it, why is magnetic recording so often overlooked or relegated to the role of a "soon-to-be obsoleted" technology? Even in home video recording, all the original excitement and speculation was based on the optical video disk—but look where we are today! Magnetic re-

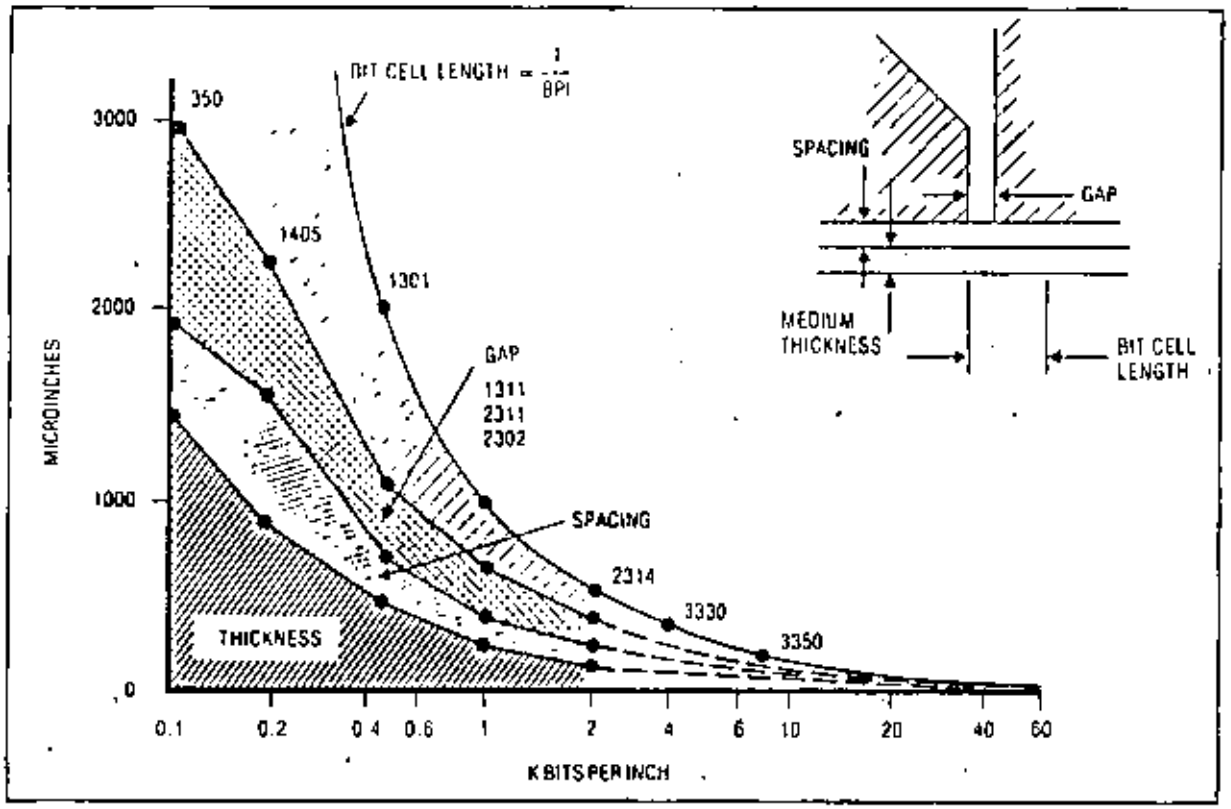


Figure 8. Bit density related to geometrical recording parameters (disk).

cording has completely taken over the home VTR market.

In an age of rapid technical change, it is hard to accept something "mature" as still being relevant. The low cost per bit of magnetic recording more than compensates for the often cited slow access time; otherwise, the technology would never have survived. In fact, one of digital magnetic recordings's oldest embodiments, half-inch tape, has defied 20 years of determined effort to bury it, either through alternative technologies or through new configurations. Actually, the security of magnetic tape seems to vary inversely with its value: the less used the archival information it stores becomes, the less is the economic justification to convert this data to another medium. In other words, time is on its side!

In a technology so old, is there anything new? In the 1970's, entirely new products have emerged:

- the floppy disk,
- the mini-floppy,
- the head-disk cartridge unit,
- the fixed disk file (for high density),
- the automatic library for both tape reels and tape cartridges, and
- digital tape cassettes and cartridges.

Technology advances continue, and we now foresee:

- thin-film heads,
- thin-film rigid disks,
- track servo information integrated with data, and
- integral microprocessor controls.

The continuing obsolescence of magnetic recording products has arisen not from alternatives but from advances in the technology itself.

Limitations

It is difficult to discuss limitations because of what one may have in mind in using this term. There are inherent physical limits related to the different storage phenomena, engineering limits related to the technical and economic feasibility of manufacturing a product, and marketplace limitations placed on the value of a storage unit in terms of other options available and additional service provided. Solid-state devices, both semiconductor and magnetic, are now being pursued by electron-beam lithography, and late 1980's chip capacities from four to eight megabits have been claimed in the press. Similar dramatic advances in magnetic recording are also foreseen. There is some feeling in the technological community that mechanical devices are too unreliable to be depended on for the storage of vital data. The facts indicate otherwise—happily so, since we have no other choice. Needless to say, we have learned to accept the continued rotation of the earth without question and

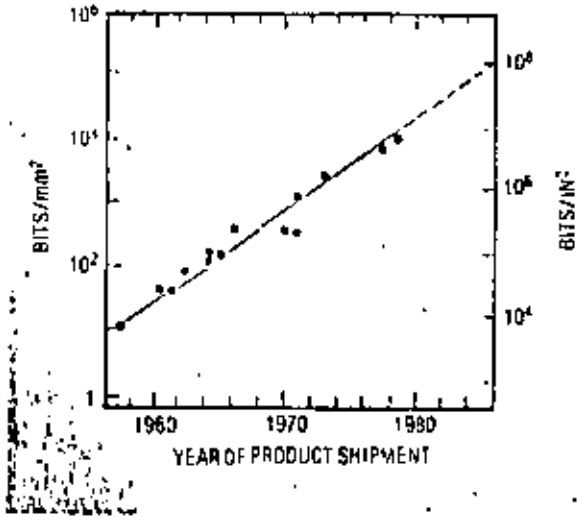


Figure 9. Disk storage density versus time.

live not wondering each day whether the sun will rise and set tomorrow.

However, evidence indicates that the growth of on-line storage capacity is greater than these technological projections. Thus, future applications for computer systems will really determine the kinds and nature of limitations that will be important. Those who desire a storage subsystem with the access speed of main memory, the capacity of a huge tape archive, and a cost so low that it is incidential compared to the overall systems cost will probably be disappointed. A hierarchy as the practical implementation of the storage function is here to stay, but the continuing combination of technological and systems progress means that the challenges will be met and will not inhibit the computer revolution.

Magnetic recording is an unglamorous workhorse that will continue to make major contributions to the growth of information processing. The significantly lower mass storage cost offered by disk, with an access time still allowing interactive usage of a data base, is the key to many advanced applications. Densities will increase up to 100 million bits per square inch (see Figure 9), and we will see capacities of several thousand megabytes per spindle with fixed disks and of up to 10 million on a floppy. The moving target will keep moving, and, yes, magnetic recording will be around through the remainder of this century. At the same time, electronic storage will continue its amazing advances. But our notion of what constitutes a reasonable storage capacity will outpace both! ■

Acknowledgment

The stimulus for this paper was the Lake Arrowhead Workshop on Data Management and Storage Hierarchies and, in particular, the unyielding persistence of the chairman, Harut Barsamian. I would like to recognize the freewheeling discussions on technology I had with John Mullinson, Leonard Laub, and Bill Doyle—although each of us had different technological biases. I therefore must assume responsibility for the impressions conveyed in this paper.

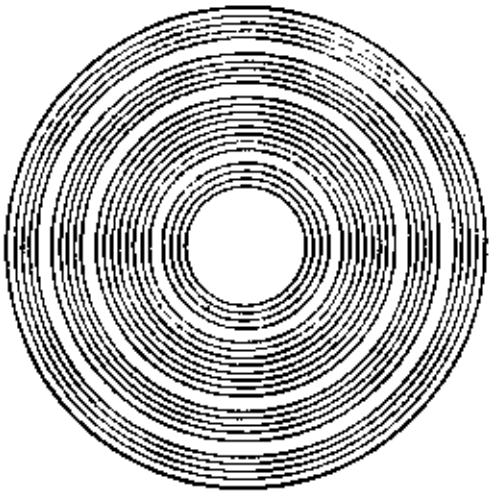
References

1. R. Bartolini, et al., "Optical Disk Systems Emerge," *IEEE Spectrum*, Vol. 15, No. 8, Aug. 1978, pp. 20-28.
2. A. S. Hoagland, "Magnetic Recording Storage," *IEEE Trans. Computers*, Vol. C-25, No. 12, Dec. 1976, pp. 1283-1288.
3. S. Puthuff, "Technical Innovations in Information Storage and Retrieval," *IEEE Trans. Magnetics*, Vol. 14, No. 4, July 1978, pp. 143-148.
4. D. Toombs, "CCD and Bubble Memories: Systems Implications," *IEEE Spectrum*, Vol. 15, No. 5, May 1978, pp. 36-39.

Albert S. Hoagland is manager of advanced recording media at the IBM San Jose Research Laboratory and president of the American Federation of Information Processing Societies. He joined IBM in San Jose in 1956, where he played a key role in the development of magnetic disk files. In 1962 he transferred to Europe as a consultant to IBM World Trade, and in 1964 joined the IBM Research Center in New York, where he was appointed director of technical planning for the research division in 1968. In 1971 he was assigned to the IBM Boulder, Colorado Laboratory to lead an interdivisional program in mass storage, and in 1976 joined the San Jose laboratory.

A Fellow and past director of the IEEE and a past president of the IEEE Computer Society, Hoagland has held many positions in the IEEE, the Computer Society, and AFIPS. He is author of the book *Digital Magnetic Recording* as well as some 25 technical publications. Hoagland holds the BSEE, MSEE, and PhD from the University of California at Berkeley.

Analytic modeling techniques permit system designers to compare memory-system alternatives quickly and easily. An example is the use of buffers—or caches—on disks.



Analysis of Memory Hierarchies for Sequential Data Access

Terry A. Welch
Sperry Research Center

Sequential data access, consisting of long sequences of accesses from consecutive logical addresses, is a major component of computer memory activity in many computer systems. An analytic study of memory hierarchy performance under sequential access, while its results will necessarily be general in nature, can nevertheless be of immediate value in evaluating design alternatives for the use of intermediate-speed memories as buffers or caches on disks. The objective of such an analysis is to provide insight into the value and type of memory-unit combinations that yield lowest cost and lowest average access times for the storage system. The strategy used in this article is to completely analyze a very simple model, containing a number of possibly oversimplified assumptions, and then to quantitatively examine the effects of relaxing each assumption individually.

The general problem of memory-hierarchy analysis concerns the proper design of various memory units having cost and access time ranges of several orders of magnitude. The objective is to select memory sizes, speeds, interconnection patterns, and data-transfer block sizes that will minimize average memory system access time for a given system cost; the strategy is to place data with the highest probability of immediate use in small, fast memory units, while holding the remainder in larger, less expensive units. This problem has received extensive study for the case of general computation involving frequent re-reference of data,[1,2,3] but the study of sequential access has been more limited.[4] An analytic approach for hierarchy analysis has been developed previously;[5] it is applied here to the sequential-access class of problems.

Sequential access is a major component in accesses to secondary storage devices such as disks. The extension of hierarchy analysis to include secondary storage is important because of the present industry trend toward including disks in virtual memory systems with automatic data transfer procedures. Also, the availability of potential "gap-filler" technologies[6] has raised interest in cache-like structures for disks. While no longer the predominant mode of secondary-storage access, sequential access is still a major storage-usage component, occurring in scientific processing, word processing, data-flow buffering, and data base activities such as searching, periodic updating and report generating. The analysis of sequential access cannot, by itself, determine optimum memory-unit parameters for general-purpose computer systems, but it can give insight into a desirable range of values for those parameters.

An analytic investigation of memory hierarchies requires the development of two reasonably independent models:

(1) An equipment model, which relates cost to access time for available memory technology;
(2) A usage model, which relates probability of access for a particular memory unit to the amount and type of data stored there, based on expected program usage patterns and inter-memory data-transfer policy.

The methodology of this study is to use an algebraic equipment model[5] that gives minimum system access time directly as a function of memory sizes and access probabilities. This permits rapid comparison, using calculus, of many configuration variations for each usage model. By these means, a reasonably efficient exploration of a multidimensional solution space is carried out.

The equipment model used here describes per-bit memory cost as inversely proportional to memory ac-

0018-9162/79/0500-0019$00.75

cess time raised to a constant fractional exponent. Variations examined for this model are the use of various exponent values and the use of individual memory units whose costs deviate from the assumed function. The usage model employed in this article assumes that all storage requests are reads in sequence from one arbitrarily long file in a two-level memory system consisting of a primary memory and a high-speed one-block buffer. The variations examined for this model are (1) access in shorter sequences, (2) double buffering, (3) intermixed accesses to multiple sequences, (4) variations in delay due to transfer times between memory units, and (5) use of more than two memory levels. This analysis of model variations serves two purposes: to assess the importance of various assumptions in the modeling process and to heighten our intuitive understanding of complex models.

An analysis technique used to achieve meaningful comparisons is to vary system parameters under a constant-cost assumption. For example, the performance improvement gained by adding a third memory level to a two-level system is more meaningfully compared to the performance of an enhanced two-level system having the same cost as the three-level system rather than to the performance of the original two-level system. The constant-cost assumption permits the derivation of desirable ratios in speed and size between memory units, and those ratios are seen to hold constant over variations in total system cost. Thus the constant-cost assumption is used to find the best memory configuration for some arbitrary cost, and the results are then scaled to fit the constraints of particular available devices or access-time requirements in real applications. The resulting configuration is then known to be the best configuration possible at that system cost. This approach does not provide a means for directly comparing suboptimum configurations; it serves only to show a bound on what can be achieved at each given cost.

The equipment model

A memory hierarchy (Figure 1) is made up of k memory units, M_i, $1 \leq i \leq k$. M_1 may be considered to be the smallest, fastest memory, while M_k is largest and slowest. Each unit is characterized by the following parameters:

- d_i cost of a data unit in M_i (e.g., dollars per byte of storage);
- c_i capacity of M_i in data units;
- t_i average access time to a data unit in M_i; more precisely, the time by which computation is delayed when waiting for M_i to respond; and
- P_i probability that a data reference requires access to M_i.

System parameters of interest are (using summations over all k memories):

- $S = \Sigma c_i d_i$ cost of the memory system, and
- $T_{acc} = \Sigma t_i P_i$ average access time of the system, assuming a single-processor system.

The access probabilities, P_i, reflect intermemory transfers as well as simple data references. For example, if any access to data in M_3 requires transfer of the data to M_1, then that access contributes to both P_3 and P_1. If data are available to the processor only through M_1, then $P_1 = 1$. Note that $\Sigma P_i \geq 1$, since one or more memory units must be accessed for each data reference. In some usage models, the P_i's are more conveniently viewed as expected values rather than as probabilities.

If one plots the unit cost versus access time of available memory units, d_i versus t_i, a monotonic decreasing relationship is observed (faster units are more expensive). It is convenient to approximate this relationship by an analytic expression that assumes a continuum of devices of all speeds. While such an assumption is usually an oversimplification, its use can produce good approximations or bounds so long as its errors are properly evaluated.

A reasonable model for the equipment function is the power function $d(t) = d_o t^{-\beta}$ for $t > 0$. Typical values of β are in the neighborhood of 0.5, which says that if memory X is four times as fast as memory Y, it should cost twice as much per bit. The justification for using a power function is that it loosely fits real data over eight orders of magnitude in the time range. Hoagland's data,[7] for example, roughly support a $\beta = 0.5$. The d_o and β parameters may take on widely different values in different applications, because of variations due to (1) system cost versus component cost, (2) system cost versus system price, (3) rapid changes in technology, (4) approximations needed to estimate average access times on nonrandom access devices such as disks, and (5) differences in measuring access time when minimizing system response time versus system throughput in multiprocessing environments. For convenience of presentation, the values $d_o = 1$ and $\beta = 0.5$ are used for much of this article.

Previous analysis of this model[5] has produced two basic results that are used extensively here. For a given set of P_i's and c_i's, the power function $d_i = d_o t_i^{-\beta}$ can be used to derive a best set of t_i's that

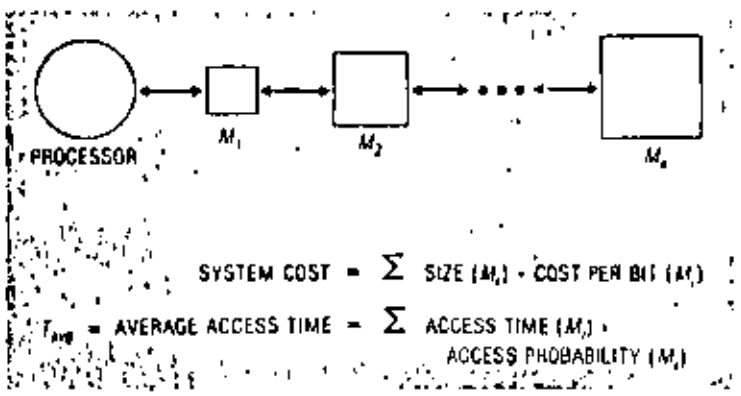


Figure 1. Memory hierarchy model.

minimize T_{avg} for a given system cost. Such an optimized system has been proven to have the following properties:

(1) $$\frac{P_i t_i}{T_{avg}} = \frac{d_i c_i}{S} \quad \text{for all } M_i.$$

that is, the percentage of system cost invested in M_i equals the percentage of system delay caused by M_i:

(2) $$T_{avg} = S^{\frac{-1}{\beta}} \left[\Sigma \left(P_i c_i^{\frac{1}{\beta}} \right)^{\frac{\beta}{\beta+1}} \right]^{\frac{\beta+1}{\beta}} T_o,$$

where T_o is an arbitrary constant. For $\beta = 0.5$, this reduces to

$$T_{avg} = S^{-2} (\Sigma P_i^{1/3} c_i^{2/3})^3 T_o.$$

This quantifies the knowledge that a low T_{avg} occurs in a multimemory system when a high portion of the accesses reference the smaller memory units. (That is, the sum is smallest when the larger P_i values are multiplied by the smaller c_i values!. In particular, the second result permits a rapid calculation of a lower bound on T_{avg} for each assignment of data to memory units.

The basic usage model

We wish to study the replacement of a single sequential-access memory unit of a given cost with a two-unit hierarchy having the same cost. This might be of interest, for example, in replacing a fast disk with a slower disk supplemented by an electronic buffer memory. The objective of the analysis is to determine the best size for the buffer, the speeds of the two memories relative to the original unit, and the resulting improvement in average access time.

For the basic model, assume that a processor continually issues requests for single units of data from sequential locations in a single file. The buffer M_1 will hold one block of B units of data, while the slow memory M_2 contains all N units of data in the system (see Figure 2). Whenever the processor requests data not in the buffer, B units are transferred from M_2, so that the next $B-1$ requests are satisfied without access to slow memory. The access time to M_2 is assumed to be determined independently of the size of B; in other words, transfer time between memories does not contribute to system delay. Basic tradeoffs appear in this simple model. If B is large, then relatively few references to M_2 are needed but M_1 is slow, while if B is small, then M_1 can be made quite fast but M_2 is frequently referenced. If M_1 is both large and fast, then it absorbs most of the system cost, leaving M_2 very slow.

Using the exponential equipment function with $\beta = 0.5$ and with $S = N$ (its units are arbitrary), property 2 above can be restated as:

$$T_{avg} \geq N^{-2} (P_1^{1/3} c_1^{2/3} + P_2^{1/3} c_2^{2/3})^3 T_o.$$

Specifically:

$P_1 = 1$, since all accesses are fulfilled out of M_1.
$c_1 = B$, since M_1 contains only one block of data.
$P_2 = 1/B$, since that is the proportion of accesses that activate M_2.
$c_2 = N$, since M_2 is constant in size regardless of c_1.

Putting these values in the T_{avg} bound, differentiating with respect to B, setting to zero, and solving for B, we get:

$$B = 0.5 N^{2/3},$$

and then

$$T_{avg} \geq 6.75 N^{-2/3} T_o,$$

where T_o is the average access time of the single-memory system. The T_{avg} of the two-memory system is expressed in terms of this T_o to show the net speed improvement relative to a single memory having the same system cost.

To illustrate the results, consider a primary memory (as in Figure 2) of $N = 10^6$ data units. (A data unit is the amount of data supplied to the processor on each access.) Then the best buffer size is $B = 5000$ units, which yields $T_{avg} = 6.75 \times 10^{-4} T_o$, showing a significant performance improvement over the single memory. Other relations derived in the earlier study[3] give buffer times for the optimized system:

$$t_i = \left(\frac{c_i T_{avg}}{P_i S T_o} \right)^{2/3} T_o$$

From this, $t_1 = 2.25 \times 10^{-4} T_o$ and $t_2 = 2.25 T_o$ are obtained directly. The slower time for t_2 relative to T_o reflects the fact that M_2 absorbs only two thirds of the system cost in the two-memory system, leaving one-third S for the smaller, faster buffer. Cost/performance balance is observed in this example, in that M_1 causes one third of the system access delay, with two thirds being caused by M_2.

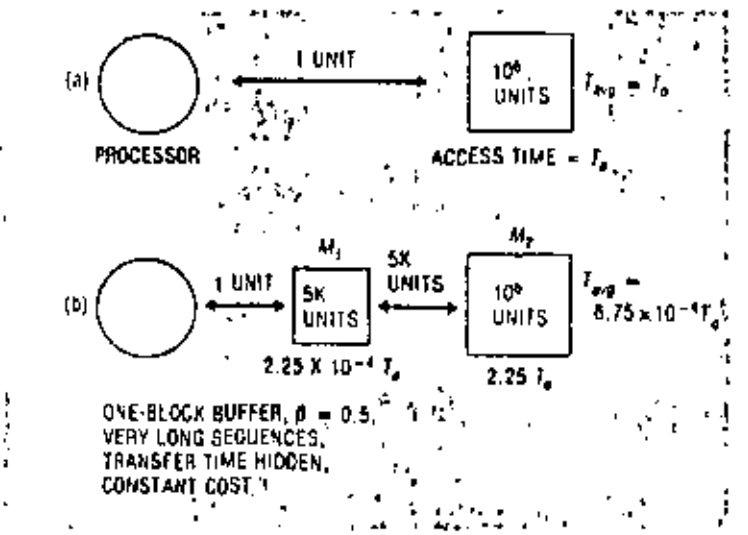


Figure 2. Two-level hierarchy example.

Variations in the equipment model

To explore various alternatives in two-memory hierarchies, the above example of $N = 10^6$ will be used throughout. In all comparisons, we assume a constant system cost.

Buffer speed sensitivity. The effect of using buffer components with various access times (t_1) is shown in Figure 3. For each component speed, the buffer size needed to minimize T_{avg} is shown, along with the resulting T_{avg} value. The sensitivity of the system to changes in t_1 can be observed in the fact that there is roughly a factor-of-four range in t_1 for which T_{avg} is within 10 percent of its minimum value. In the right-hand portion of the curve, T_{avg} increases as the buffer becomes slower and buffer access delay becomes the predominant component in T_{avg}. In the left-hand portion of the curve, cost constraints on the buffer cause it to become smaller as the component speed and cost increase, thereby causing more frequent accesses from M_2 and giving a slower T_{avg}. An interesting property of this tradeoff relationship is that a constant fraction of system cost is maintained in the buffer for all values of t_1. That is, for any t_1 value, the lowest T_{avg} is achieved when buffer size is such that the cost of M_1 is β times the cost of M_2, namely, $c_1 d_1^{-\beta} = \beta c_2 d_2$.

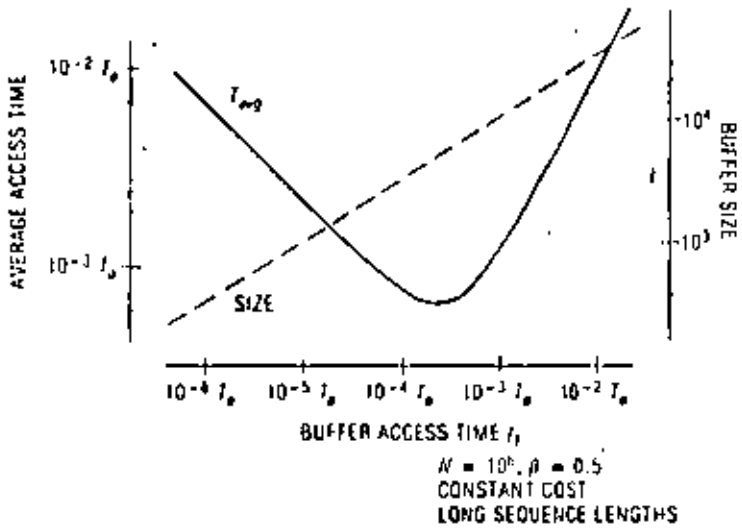


Figure 3. Effects of changes in buffer access time.

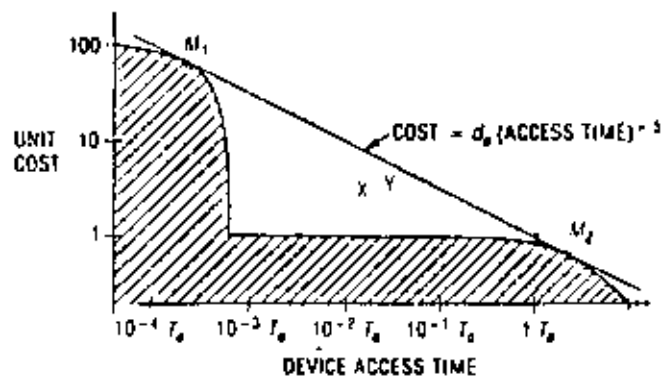


Figure 4. Range of equipment that can achieve example system performance.

Table 1.
Effects on average system access time (T_{avg}) of several values of the equipment function (β) for the two-memory system of Figure 2.

β	T_{avg}
0.7	$15.30 \cdot 10^{-4} T_a$
0.6	$10.40 \cdot 10^{-4} T_a$
0.5	$6.75 \cdot 10^{-4} T_a$
0.4	$4.20 \cdot 10^{-4} T_a$
0.3	$2.52 \cdot 10^{-4} T_a$

Equipment function sensitivity. The effects of variations in the exponent β are shown in Table 1. This shows the minimum T_{avg} for the $N = 10^6$ two-memory example (Figure 2) using various values of β. The range of values given covers the full range generally observed in memory technology comparisons. This indicates that the use of $\beta = 0.5$ will give results that are within a factor of two or three of the results obtained with extreme equipment functions. The formal relationship is:

$$T_{avg} \geq (1 + 1/\beta)(\beta + 1)^{1/\beta} N^{\frac{-1}{1+\beta}}$$

Alternative memory availability. If a memory unit were available that offered better price/performance than the memories described by a $d(t) = d_0 t^{-\beta}$ function, it might provide a better configuration. For the example two-memory system, Figure 4 shows the range of such units that are of value. Only memories falling within the shaded area could produce a faster system at the same cost as the original system. This indicates that minor variations in equipment availability, relative to the power-function model, would not dramatically change the character of the results studied here. For example, if some other memory unit Y, midway between M_1 and M_2 in cost and speed, were somewhat better than the $d(t) = d_0 t^{-\beta}$ curve, it still would not offer a better buffer configuration than the M_1 unit.

Variations in the usage model

Continuing to use the $N = 10^6$ and $\beta = 0.5$ example, we can examine several variations in the usage model. The intent here is to illustrate the effect of application changes rather than to test the original model, as was done in evaluating variations to the equipment model.

Variable sequence lengths. The basic model of Figure 1 assumes that all accesses come in one long address sequence, whereas in real systems a variety of finite-length sequences are used. Let L be the

average number of references in a sequence, with actual sequence lengths randomly distributed such that on any particular access there is a $1/L$ chance of starting a new sequence. Assume that each sequence starts at a new location not related to the previous sequence addresses, so that starting a new sequence requires an access to M_2 regardless of the size of M_1. Then the expected number of accesses to M_2 is $P_2 = (L - L(1-1/L)^B)^{-1}$. The other parameters remain at $P_1 = 1$, $c_1 = B$ and $c_2 = N$. The resulting equation for best block size B has no simple solution, but a hand calculator is sufficient to develop the sample results shown in Table 2 and summarized in Figure 5. Note that for short average block lengths it is economically useful to have a buffer significantly bigger than L, which would not be immediately obvious without such an analysis. Small values of L also have the effect of flattening the curve shown in Figure 3, so that T_{avg} becomes less sensitive to changes in B and t_1 as L becomes smaller.

Variable transfer time. The basic model assumes that M_2's access time did not depend on block size B, which implies that the transfer time was "hidden" behind computation time. That is, the immediately needed unit of data is transferred from M_2 first, and computation can proceed during the remaining transfers, with interlaced buffer-cycle allocations. In some systems, however, the transfer would interfere with computation, and that effect is modeled here by assuming an extra t_1 cycle for each data unit moved between M_2 and M_1. This is seen to increase the expected number of accesses to M_1 by the number of data units transferred from M_2, so that $P_1 = 1 + P_2 B$. The other parameters used are those for variable-length sequences: $P_2 = (L - L(1-1/L)^B)^{-1}$, $c_1 = B$, and $c_2 = N$. Again, the solution equation is not simple, giving sample values as shown in Table 3. The inclusion of transfer time is seen to have a small effect on T_{avg} but a somewhat larger effect on choice of buffer size and speed. This occurs because large block transfers are penalized by added transfer time; a smaller, faster buffer is therefore appropriate, as one would expect.

Double buffering. A common way to implement sequential accesses is to divide the buffer into two equal blocks and have computation proceed out of one block while the next block is being fetched from main memory. Interesting tradeoffs occur only in the case in which access sequences are of variable length and in which computation time on a block is less than t_2 (namely, when the blocks in M_1 are used faster than M_2 can fill them up.) This analysis requires the introduction of a processor-speed variable—namely, the delay between processor accesses to the memory system. The model used here assumes that once the processor receives a unit of data from the memory, it will not produce another request for t_1 time units (meaning that the processor is tuned to be about as fast as the buffer memory). The model which assumes that processing time within a block is shorter than t_2 is then valid only for $2t_1 \cdot B \ll t_2$. For this case, all accesses to M_2 are viewed as normal access-time delays: $P_2 = 1/L\,(1-(1-1/L)^B)^{-1}$. However, since accesses to M_1 that are to full blocks in a sequence are carried out in parallel with a concurrent M_2 access to a later block in that sequence, these M_1 accesses do not cause system delay and are not counted in P_1. The only contribution to P_1 is from accesses to fragments of a sequence that will not use the next buffer and so are not concurrent with any useful activity in M_2: $P_1 =$

Table 2.
Effects of variable sequence lengths ($N = 10^6$ data units).

AVERAGE SEQUENCE LENGTH (L)	BUFFER SIZE (B)	BUFFER ACCESS TIME (t_1)	SYSTEM ACCESS TIME (T_{avg})	PERCENT OF SYSTEM COST IN BUFFER
>>10K	5000	$2.25 \cdot 10^{-4} T_a$	$0.0007 T_a$	33
10K	4290	$2.25 \cdot 10^{-4} T_a$	$0.0008 T_a$	29
5K	3770	$2.26 \cdot 10^{-4} T_a$	$0.0009 T_a$	25
3K	3258	$2.27 \cdot 10^{-4} T_a$	$0.0010 T_a$	22
1K	2030	$2.35 \cdot 10^{-4} T_a$	$0.0018 T_a$	13
500	1360	$2.45 \cdot 10^{-4} T_a$	$0.0028 T_a$	9
256	875	$2.60 \cdot 10^{-4} T_a$	$0.0048 T_a$	5
100	440	$2.85 \cdot 10^{-4} T_a$	$0.0110 T_a$	3

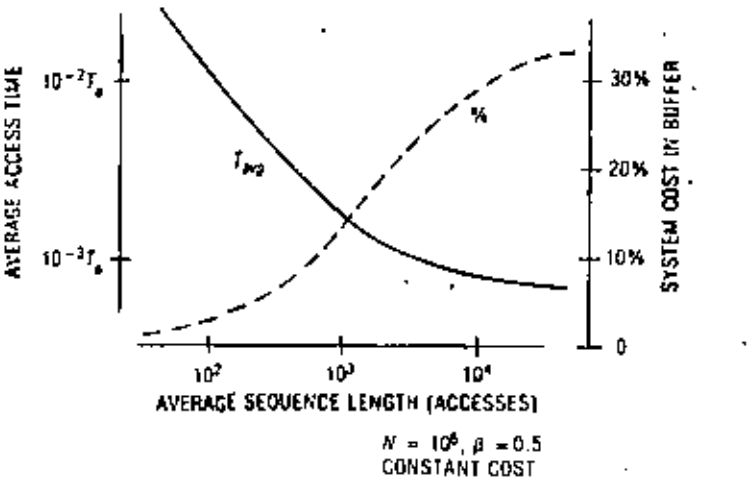


Figure 5. Effects of various sequence lengths.

Table 3.
Effects of variable sequence lengths, transfer time included ($N = 10^6$ data units).

AVERAGE SEQUENCE LENGTH (L)	BUFFER SIZE (B)	BUFFER ACCESS TIME (t_1)	SYSTEM ACCESS TIME (T_{avg})
>>10K	4000	$1.42 \cdot 10^{-4} T_a$	$0.0008 T_a$
10K	3300	$1.31 \cdot 10^{-4} T_a$	$0.0010 T_a$
3K	2500	$1.20 \cdot 10^{-4} T_a$	$0.0013 T_a$
1K	1500	$1.07 \cdot 10^{-4} T_a$	$0.0021 T_a$
500	1000	$0.96 \cdot 10^{-4} T_a$	$0.0032 T_a$
100	353	$0.92 \cdot 10^{-4} T_a$	$0.0115 T_a$

$1 - B(1 - 1/L)^B P_2$. Since two blocks of buffer are used, $c_1 = 2B$.

The results of this model show that $2t_1 \cdot B \leq t_2$ is valid only for sequences of length $L \leq 10^3$ (for the $N = 10^6$ case). For these cases, an interesting result occurs: double buffering always causes a slight increase in T_{avg} and is therefore not of value. This says that, for shorter access sequences that do not frequently utilize the subsequent M_2 data transfers, the advantage of starting up later M_2 accesses in parallel with computation from M_1 is outweighed by the cost lost in the extra buffer space. For the $L > 10^3$ case, memory times are determined just by setting $2t_1 \cdot B = t_2$. In this case, some saving in T_{avg} is realized—up to almost a factor of two for very large L.

Multiple files. The basic model assumed that only a single address sequence was being carried on at one time. In general-purpose computer systems, each user program will in fact be intermixing accesses from several files, and several such programs may be in operation at once. If n sequences are active concurrently, then $c_1 = nB$ (for the single-buffer case). This has an easily calculated effect, in that resultant block sizes and access times are determined as if N were reduced to N/n. For $L >> B$, all average access times are therefore multiplied by $n^{2/3}$. This quantifies the observation that an increased degree of multiprogramming causes each program to run somewhat slower, owing to less intensive use of buffer memory and, consequently, less effective use of the memory hierarchy.

Multiple memories. The model allows us to observe the effects of using three (or more) levels of memories instead of two. Under the original assumptions of a single buffer and a single long access sequence, $c_1 = B_1$, $c_2 = B_2$, $c_3 = N$, $P_1 = 1$, $P_2 = 1/B_1$, and $P_3 = 1/B_2$, with the results shown in Table 4. The table also shows sample results for more populous hierarchies. Note that adding each additional level of memory gives a much smaller performance improvement than adding the previous one. These results must be interpreted with care, however, because a processor built with a three-memory system might be designed to access smaller units of data. This would have the effect of changing N, because memory size is calibrated in the units of processor request size, and the larger N would give more effective use of the extra memory levels.

Table 4.
Effects on system timing of using multiple memory levels ($N = 10^6$ data units).

NO. OF LEVELS	MEMORY UNIT	MEMORY SIZE	ACCESS TIME	FRACTION OF SYSTEM COST	T_{avg}
1	M_1	$c_1 = 10^6$	$t_1 = T_0$	1	T_0
2	M_2	$c_2 = 10^6$	$t_2 = 2.25\ T_0$	2/3	$6.75 \cdot 10^{-4} T_0$
	M_1	$c_1 = 5 \cdot 10^3$	$t_1 = 2.25 \cdot 10^{-4} T_0$	1/3	
3	M_3	$c_3 = 10^6$	$t_3 = 3T_0$	4/7	$1.3\ 10^{-4} T_0$
	M_2	$c_2 = 42 \cdot 10^3$	$t_2 = 22 \cdot 10^{-3} T_0$	2/7	
	M_1	$c_1 = 607$	$t_1 = 1.8 \cdot 10^{-5} T_0$	1/7	
4					$0.76 \cdot 10^{-4} T_0$
5					$0.65 \cdot 10^{-4} T_0$

Application example

Let us explore the use of a buffer memory as a cache on a disk for system applications primarily involving sequential access. Assume a disk with 8000 tracks, and assume that each processor request is for 4000 bytes, or one-eighth of a track; then $N = 64{,}000$. Assume that $n = 4$ files are open on the disk concurrently. In this example it is desirable to add cost to the disk system to improve performance; t_2 will therefore be held fixed, and cost S will be increased, but the relative memory-unit speeds are still those derived by the previous analysis. For long sequences, $B = 0.5\ (N/n)^{2/3}$, $t_1 = (3/2)^2\ (N/n)^{-2/3}\ T_0$, and $t_2 = (3/2)^2\ T_0$. This indicates a block size of 40 tracks and a buffer size of 160 tracks. This is not inconsistent with the empirical observations of Smith.[4] At a disk access time of $t_2 = 30$ milliseconds, then $t_1 = 47$ microseconds and $T_{avg} = 0.14$ milliseconds. This solution involves adding 50 percent in cost to the memory unit to achieve a factor-of-214 access-time improvement for sequential accesses.

However, the block size of 40 tracks is unrealistically large, since most access sequences are not that long. The numbers given must be viewed as bounds from the ideal sequential access system; any imperfections will favor a smaller buffer and higher T_{avg}. Perhaps a more reasonable system would use a block size of 20 tracks (one cylinder), which would be a good size for average access-sequence lengths of around $L = 12$ tracks. This would employ a buffer of 80 tracks with $t_1 = 75$ microseconds, giving T_{avg} of 0.45 milliseconds at 20 percent extra cost.

What do these numbers mean to the system designer? The derived configuration will give the best performance of any system having the same cost, and the ratio of unit speeds will give the best performance for any specified cost. Consider the case of long sequence lengths, where 50 percent extra cost was added to a disk as a buffer. The configuration that has a block size of 40 tracks in a 47-microsecond buffer can be shown to give the best performance of any possible buffer system adding 50 percent to the disk cost, provided components are used that are not better than the $\beta = 0.5$ power function indicates. Like most other bounds, it probably cannot be implemented directly, but it does give design insight and a benchmark by which to evaluate other designs. If a system is desired at less than 50 percent extra cost, the use of a smaller or slower buffer on the same disk would still give good performance. This analysis, however, indicates that a better system might be available by finding a somewhat slower, cheaper disk and holding the buffer at 50 percent of disk cost. Conversely, putting more than 50 percent cost into the buffer would

achieve better performance, but probably not as much performance improvement as could be achieved by developing a faster disk (for example, by improving the head-to-track ratio). Thus, the derived buffer configuration provides a guide for resource allocation as well as a bound on attainable performance.

The unfortunate result of this analysis is the clear indication that, since buffer size should depend very much on average sequence length, no one buffer configuration will provide a best solution for all applications.

The above analysis is incomplete because it ignores possibly desirable changes in the size of the basic data unit. An alternative model is to include the processor's main memory in the analysis by assuming a data unit of one word (four bytes). Assuming 8000 words per track, and assuming a system of eight disk drives connected to one processor, $N = 2^{29}$. If four files are open per drive, the system has $n = 2^5$ concurrent access sequences. A buffer size of $c_1 = 10^6$ words, transferred in 32K-word blocks, with access time $t_1 = 0.7$ microseconds, is then indicated. This is surprisingly close to processor main memory specifications, with some variances attributable to the use of a shorter-than-infinite sequence length.

To study disk buffers, consider a three-level configuration for the problem shown in Figure 6. This produces a slow buffer of $15 \cdot 10^6$ words at 100 microseconds, loaded in blocks of 475K words, and a fast buffer of 96K words at 30 nanoseconds, loaded in blocks of 3K words. This says that a disk buffer should load directly into the processor cache memory in sequential access operations. Again, the block sizes are excessive, which means that a short sequence analysis is needed for each application. What does this result imply about placing another memory (such as a conventional main memory) between the disk buffer and the processor cache? The model results do not imply that no more performance can be gained by this approach; they only imply that any performance gained may be relatively expensive compared to other alternatives.

Of course, the above results cannot be used directly to design processor memory systems because other data reference patterns are encountered. Specifically, a tendency to reaccess some data, as opposed to the assumed accessing of each word just once, would lead to smaller block sizes. The analysis given does serve the purpose of showing bounds on block sizes and buffer access times and thus yields insight, even if not precise solutions.

Conclusion

This study illustrates the advantages and limitations of analytic modeling. The results derived here cannot compete with those of simulation or actual design in accuracy. This approach does, however, produce the desired ability to rapidly compare various system configurations in order to identify promising directions for more detailed investigation. An equally important benefit is the identification of systems parameters and of their relative importance in the analysis.

These results clearly show the value of a hierarchy of memory units in a sequential data stream, within the constraints of a typical system in which the various resources are being shared among different activities. The results are not surprising, but neither were they fully anticipated. The value and role of a hierarchy are not as intuitively clear for sequential accesses as they are for accesses that are well described by a LRU stack model. The analysis provides insight into the value of buffering, which permits very large block transfers from slow devices and fast access to smaller data units.

The principal results of this analysis can be summarized in terms of the parameters that affect buffer specification selection. Buffer access time and size are reasonably sensitive to N, and, therefore, sensitive to the size of the accessed data unit and to the degree of multiprogramming, n. Buffer speed is not influenced by average sequence length L, while buffer size is directly affected. In particular, desirable buffer size is proportional to L for small values of L and is determined by N for large values of L, with the breakpoint determined by the long-sequence block size. Also of interest is that the fraction of system cost to be invested in the buffer is not a function of N but is very much a function of L. These basic relationships are expected to apply in many systems that lie outside the range of the assumed model.

Recommendations for desirable disk buffer characteristics can be based in part on the examples developed here. My conjecture is that a buffer will have to provide an access time (within a factor of two) of 100 microseconds at a production cost of 0.005 cents/bit to be commercially successful in new memory hierarchies in 1980. This is based on present estimates of 5×10^{-7} seconds access time at 0.06 cents/bit system production cost for 64K RAMs and

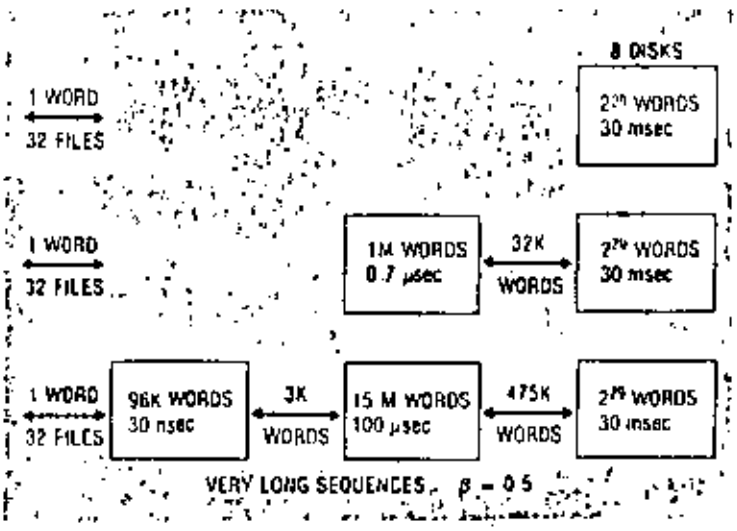


Figure 6. Disk system example.

3×10^{-2} seconds access time at 0.0003 cents/bit cost for large-capacity disks. This conjecture is based on the feeling that new memory devices will have to be as good as the power function established by RAMs and disks in order to achieve a high volume market. It also is based on the feeling that desirable buffer access times for sequential access are not in conflict with the values needed for other access patterns. ■

Acknowledgment

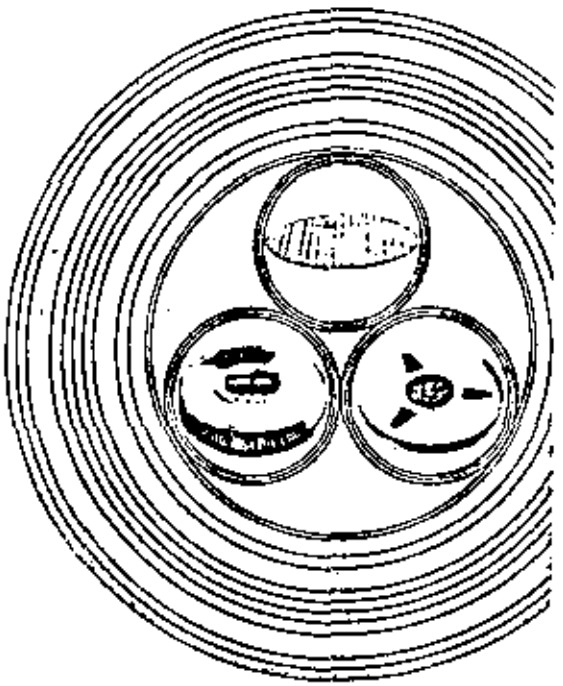
I appreciate the aid and encouragement of Harut Barsamian during the development of this paper.

References

1. R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM Systems Journal*, Vol. 9, No. 2, 1970, pp. 78-117.
2. P. A. W. Lewis and G. S. Shedler, "Empirically Derived Micromodels for Sequences of Page Exceptions," *IBM Journal Res. and Dev.*, Vol. 17, No. 2, Mar. 1973, pp. 86-100.
3. R. Turner and B. Strecker, "Use of the LRU Stack Depth Distribution for Simulation of Paging Behavior," *CACM*, Vol. 20, No. 11, Nov. 1977, pp. 795-798.
4. A. J. Smith, "On the Effectiveness of Buffered and Multiple Arm Disks," *Proc. Fifth Symposium on Computer Architecture*, Apr. 1978, pp. 242-248.
5. T. A. Welch, "Memory Hierarchy Configuration Analysis," *IEEE Trans. Computers*, Vol. C-27, No. 5, May 1978, pp. 408-413.
6. A. S. Hoagland, "Storage Technology: Capabilities and Limitations," *Computer*, Vol. 12, No. 5, May 1979, pp. 12-18.
7. S. L. Rege, "Cost, Performance and Size Trade-Offs for Different Levels in a Memory Hierarchy," *Computer*, Vol. 9, No. 4, Apr. 1976, pp. 43-51.

Terry A. Welch is manager of the Computer Architecture Department at the Sperry Research Center, Sudbury, Mass. His recent research includes memory systems analysis and processor design. Before joining Sperry in 1976, he was assistant professor of electrical engineering and computer sciences at the University of Texas at Austin for five years.

Welch received the SB, SM, and PhD degrees in electrical engineering from the Massachusetts Institute of Technology in 1960, 1962, and 1971, respectively. He has been active in Computer Society activities as chairman of the Central Texas Chapter, and is a member of ACM.



Tomorrow's data base systems will be organized around a common architecture, data distribution techniques, and new specialized processors.

Current Trends in Data Base Systems

G. A. Champine
Sperry Univac

The current status of data base systems can be characterized by two significant trends: rapidly increasing user acceptance, and rapidly improving technology at the logical, physical, and architectural levels. These two trends are closely coupled. Increased user acceptance has come about, in part, because the price of direct access mass storage has fallen by a factor of ten in the last eight years and is currently falling by a factor of two every 30-36 months. The availability of low-cost direct access mass storage has encouraged users to move away from tape-oriented sequential files into structures that support direct access efficiently. A recent survey has indicated that about half of the medium and large systems now being procured will have data base applications.

Conversely, the large market for data base systems has spurred research and development to provide increasingly better technology. Of course, fundamental to this "positive feedback" situation are the basic value of data base systems to the end user and the absence of substantive barriers to further technological improvement.

Even though the overall cost/performance of systems is improving by a factor of two every 4-5 years in current dollars, the needs of computer users are growing even more rapidly. This trend is leading to considerable interest in computer architectures that are more efficient in information storage and retrieval applications. Essentially all of the architectural design activity in information storage and retrieval is directed at data base systems. Data base systems were developed to overcome a number of shortcomings in existing file management systems. Advantages of data base systems can be listed as follows:

- Storage design is independent of specific applications.
- Explicit data definition is independent of application programs.
- Users need not know data formats or physical storage structures.
- Integrity assurance is independent of application programs.
- Recovery is independent of application programs.
- Keeping a single copy of the data eliminates the redundancy and inconsistency of multiple files.

Although a number of approaches to data base systems exist, three data models appear to be gaining dominance: the network model, as specified in the Codasyl data design language and *Journal of Development*; the relational model; and the hierarchical model.

The scope of this paper is to consider the technological advances in four categories related to data base systems:

- mass storage hardware,
- user interfaces,
- distributed data base systems, and
- data base computers.

The history and benefits of data base systems are by now well known and need not be reviewed again here. However, because of the rapid advance of technology in this area, an overview of current research activities and technology trends is in order.

0018-9162/78/0500-0027$00.75 © 1978 IEEE

Mass storage hardware

Current storage systems include a hierarchy composed of random access (or main) storage, direct access storage, and sequential access storage.

Main storage. Main storage is generally ignored in discussions of mass storage, but main-storage technology has a major influence on how other elements of the storage hierarchy are used.

Main storage provides the necessary buffer area to operate direct access or sequential access storage, and is provided today in capacities that would have qualified for mass storage only one decade ago. Whereas in the early 1970's main storage units of more than four megabytes were rare in large systems, main storage units of less than four megabytes are now rare on the same systems. As main storage has become cheaper, the optimum system balance has shifted toward larger main stores; as a result, processor utilization has increased dramatically from the 50 percent to 60 percent common in the early 1970's to the 85 percent to 95 percent common today. The increased main store size has also decreased the amount of I/O performed because more data are kept in main storage rather than moved to a lower level in the storage hierarchy. These data can be retained in the main store in the form of hardware or software pages or in the form of complete files or data bases.

Figure 1 shows how the cost of main storage has fallen as the technology has moved to progressively higher levels of chip integration for dynamic MOS. Overall, the price has fallen from 1.0 cent/bit in 1970 using 1K chips to 0.1 cent/bit in 1978 using the current industry standard 16K chip. The general pattern has been to quadruple the number of bits on a chip every three years; this has cut the price approximately in half every three years (a nonlinear cost reduction because cabinets, cables, power supplies, and interface logic have much more constant costs). Samples of 65K chips are now available, and products containing these chips should be in mass production by 1980, thus continuing the trend.

Current improvements in photolithography suggest that the 262K chip will be feasible, with samples likely in 1982, and that a 1 megabit chip may even be possible later. At these levels of integration, the cost of main storage is mostly in the interface and in support equipment and is almost independent of the size of the store. The low cost of these devices is leading to ever-larger main stores in all classes of computer systems.

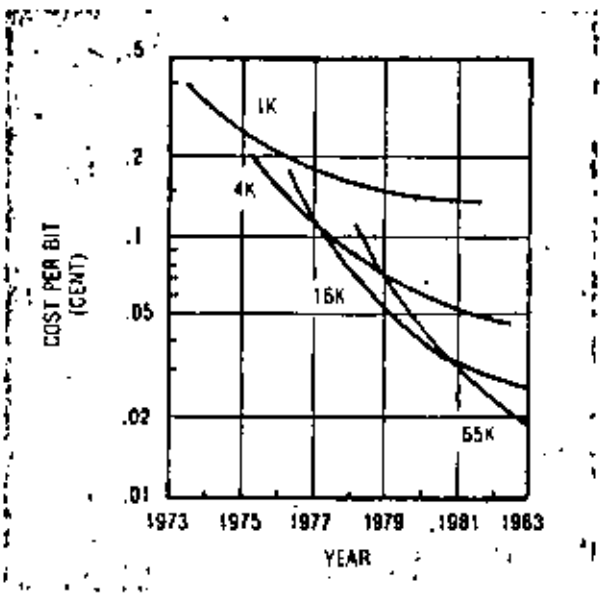


Figure 1. Trends in costs of main storage using dynamic MOS ICs indicate that the number of bits per chip has quadrupled every three years, cutting the price in half during each such period.

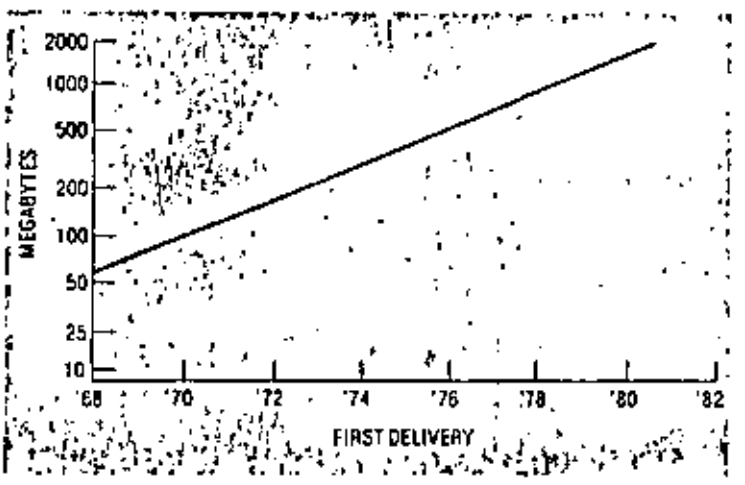


Figure 2. The capacity of moving-head disks has increased steadily since 1965, due primarily to increases in recording density.

Direct access storage. The requirement for direct access storage has traditionally been satisfied by magnetic disk, with either fixed or moving heads. More recently, two new solid-state direct access technologies have appeared: CCDs and bubbles. Optical storage devices using lasers have been proposed and some prototypes have been developed, but it does not appear likely that a viable optically-based product will be available in the near future.

Disks. The capacity of magnetic disks has increased rapidly because of increases in recording density. Storage capacity for a state-of-the-art moving-head disk, as shown in Figure 2, increased from about 30 megabytes in 1965 to 600 megabytes in 1978. Recording density has increased from 300 bits/sq. cm. (2000 bits/sq. in.) in the early 1960's to 1 million bits/sq. cm. (6 million bits/sq. in.) currently. The recording density is determined by the thickness of the oxide coating on the disk, the flying height of the head, and the gap in the head. These three dimensions must be roughly similar in size and are currently in the 0.6-micron (25-microinch) region. They will be reduced by new techniques to provide increased recording density. The magnetic coating on the disk will be upgraded to support the increased resolution, either by better control of granule characteristics and orientation or by the use of pure-metal-plated sur-

faces. Hoagland discusses this topic in much more detail elsewhere in this issue (p. 12).

The technology for read/write heads continues to receive considerable attention, with particular emphasis on thin-film heads. Thin-film heads offer both lower cost and higher performance: lower cost because they are made by batch fabrication techniques and higher performance because they offer more precise control over dimensionality.

The result of these continued advances is that the cost reduction per byte by a factor of two every 30 months will continue for at least several years. Disk capacity should double again to 700 megabytes in 1980 and to more than one gigabyte by 1982. The cost per on-line byte stored will decrease in direct proportion.

Sequential access storage. Bubble and charge-coupled devices, both sequential access technologies, have also increased rapidly in density. Bubble devices with 250,000 bits per chip have been announced and there is strong evidence that a 1-megabit chip will be available in late 1979 or 1980. However, as bubble chips have gotten larger and cheaper per bit, they have gotten slower, current devices having a 7-msec access time. It appears now that they will not be used as mass storage on large mainframes, but will probably be used on small systems. Recent improvements by Bell Labs avoid the coils required to obtain the rotating magnetic field. This field is required to shift the bubbles along the lattice structure. This new development should improve the shift rate by a factor of five and provide an access time approaching that of CCDs.

Charge-coupled storage is now available in 65K chips, with good promise for 262K chips in 1980. Access times for this technology are in the half-millisecond range. The historical 18-24 month lead of CCDs over RAMs may make the former attractive in high-speed swapping stores and as disk caches for large mainframes. However, the rapid advance of RAMs makes this a close race.

The user interface

The user interface is the logical interface into a data base system. Among the many research activities in this area, the following appear to be especially significant:

- data models,
- data base access approaches, and
- common data base architecture supporting multiple views of data, schema-to-schema translation and schema, subschema, and storage schema separation.

Data models. Among the several dozen user interfaces proposed for data base systems, the two principal approaches are the relational data model and the network data model. The network model is the more mature, having been developed in the period around 1970. This model, as characterized by the Codasyl data base task group specification, provides relatively high performance but requires the user to specify storage structures, access paths, and data structures in considerable detail. The network approach is also characterized by inflexibility, in that access paths not predefined at data base load time can never be used. The storage structures, generally constructed of pointers to linked lists, tend to be quite complex.

In contrast, the relational approach generally shields the user from the complexity of storage structures, data structures, and access paths. Access paths need not be predefined (although options may allow the user to predefine them). Without predefined access paths, however, performance can suffer badly. The storage and data structures are very simple; they are treated as a series of one-level records. A much more detailed description is provided by Astrahan et al. elsewhere in this issue (p. 42).

Although the network model dominated data base technology for almost a decade, interest is now growing in the relational model, whose attractive features include a simple yet powerful interface and the ability to process ad hoc queries (i.e., those without predefined access paths). Although the relational approach may be less efficient, the declining cost of hardware, coupled with the rising cost of manpower and the very real value of fast response to ad hoc queries, makes relational data models increasingly attractive. In addition, the associative storage devices described below offer the potential of greatly improving the efficiency and therefore the performance of relational systems.

Both the network and the relational data models exist as supported products, although the network approach has a large existing user base and is gaining momentum for the near term. The network approach may be characterized as a host-language (usually Cobol) embedded network system: it supports index sequential, hash, direct, and set-location access modes. For example, a record about a city could be accessed through the state in which it is located (via set), alphabetically (index sequential), or through the country in which it is located (via set). The advantage of the network approach is high performance, because all of the access paths can be defined and built (by using pointers) at data base creation time. The disadvantage is that it is a relatively low-level language and involves the user in storage management and detailed record access. It is also limited to queries that can be satisfied by the predefined access paths.

In contrast, the relational data model approach is a high-level data retrieval and manipulation language that shields the user from data formats, access methods, and storage management. Access paths do not have to be predefined. A typical query might look something like:

"list employees where department = design and skill = engineer"

The lack of predefined physical access paths means that relational data bases must be exhaustively searched to satisfy a query. Since this searching is very slow on conventional computers, the user is often allowed to optionally specify a prior access path, even in relational systems, to obtain acceptable

performance. (One way to obtain the benefits of a relational user interface without the drawbacks of predefined access paths is to provide special-purpose associative processing hardware to speed table searching, as described below.)

A third model is the hierarchical approach, which is a subset of the network approach in that groups of records (structures) can be addressed by only one logical path.

Hierarchical, network, and relational data bases. The three popular approaches to data base access, then, can be classified as hierarchical, network, and relational.

In the hierarchical approach, records with some characteristic in common are grouped into sets. For example, a data base for a university curriculum might contain records of information on courses in a department; these records might be grouped into sets by department. The same data base might contain records of information on students; these might be grouped into sets by advisor. A complete structure might be as shown in Figure 3. Here, each "department" record is the owner of a "teacher" set and a "courses" set. The "department" set is in turn owned by the "curriculum" record. If one wanted to access data about a given course (for example statistics) within a given department (for example mathematics), the set belonging to the "curriculum" record would be searched until the "department" record was found. The "course" set belonging to the "department" record would be searched until the "mathematics" record was found. The set belonging to the "mathematics" record would be searched until the "statistics" record was found. The process of following logical paths from owner sets to member sets, and to records within member sets, is called "navigation."

A network data base is a generalization of a hierarchical data base. A hierarchical data base is restricted to a single owner for each member set; however, a network data base can have multiple owners for a member, as is shown in Figure 4. Here, the "students" set can be reached through either departments or advisors.

In a relational data base, the information is stored in the form of "relations," with each entry in the relation called a "tuple" or record. Our example of a college data base could include the relations shown in Figure 5. To find the name and location of the teacher of statistics, the course-teacher relation is searched, to find the "statistics" record from which the teacher name "Lindgren" is obtained. Then the course-location relation is searched to find the "statistics" record from which the location "Science 26" is obtained. The answer from the data base then is "Lindgren, Science 26," obtained from the record value "statistics" held in common.

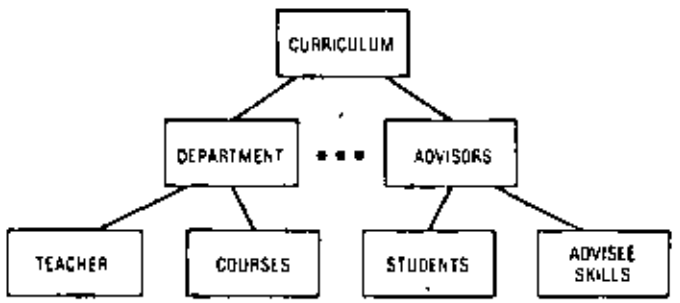


Figure 3. In a hierarchical data base, each set of records (e.g., on students) is "owned" by a single superset (e.g., advisors).

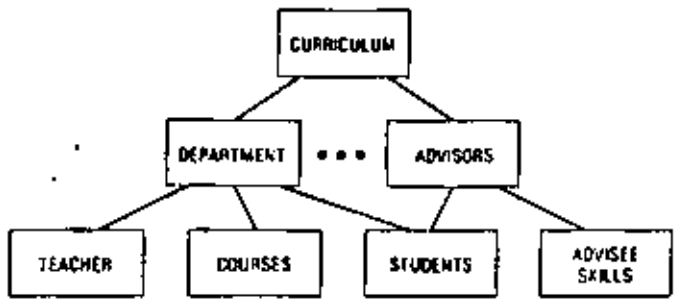


Figure 4. In a network data base, a set of records (e.g., students) may be "owned" by more than one higher classification; in this illustration, the "students" set is owned by both "department" and "advisors".

Toward a common data base architecture. A movement toward unification and simplification of the various approaches to data base systems is now in progress. This movement is similar to the unification and simplification that has taken place in computer languages in the last few years. One example of work in this direction is the common data base architecture[5] which provides a unified basis for the various approaches to data base systems in use today. In addition to providing a unified basis for understanding and description, the common data base architecture makes possible the following:

- demonstration that features in network and relational systems are equivalent to each other,
- translation of data definition and data manipulation statements from one data model to another,
- translation from one data model schema to another,
- superposition of the data description and data manipulation constructs from one data model on the schema constructs of another,
- support of a single data structure in a system, and
- translation of data manipulation commands in a heterogeneous distributed system.

The methodology of the common data base architecture is to expose the commonality that exists among different data models and also at different levels within the same data model.

As an example of the equivalence in functionality among different data models, consider how inter-record relationships are handled in the network and relational data models. In the network model, inter-rec-

ord relationships are explicitly defined by the declaration of set relations in the schema. The set relation logically chains together those records having some common relationship: for example, all employees in a department could be logically associated by defining them to be members of a set occurrence. In the relational model, inter-record relationships are defined implicitly by having the same value in a field. Each employee record, for example, could contain a department field, and all employees in the same department would have the same department number in that field. Thus, the two models provide the same functionality, but by different mechanisms. If it were desired to translate from a network to a relational model, the linked-list pointer in the employee record would be changed to be a department field, and the values in the field would be copied from the owner of the set. If the relational model were to be translated to the network model, the department value would be inserted in the set-owner record, and the department field would be changed to be the linked-list pointer. On a higher level, the statement type "set" in network models corresponds to common field names for records in a relational model.

More explicit examples of the correspondence between constructs in the network and relational data models are as follows:

NETWORK	RELATIONAL
record type	table (relation)
item	column (component)
record occurrence	row (tuple)
unique record key	candidate key
principal key	primary key

Support of multiple views of data. In the current implementation of data base systems on von Neumann computers, network data models are supported by pointers, and relational data models are supported by table searching. There appears to be little or no research in progress to design architectures that are better at following pointers, but there is considerable research in the area of faster searching of tables. This is described in more detail below, in the section on trends in data base architecture.

In the network model, associations among records are made by pointers. For example, a department record could contain the origin of a pointer chain to the members of the department. In the relational model, associations among records are made by the same value of a field. For example, a department record could contain, among other things, the department number. The members of that department would also contain the department number in each of their records.

The pointers and the common key values contain the same information and can be used interchangeably. If a relational model were built with pointers, the pointers would be used to associate records with a common field value and would, in fact, replace the field value, which would now only appear once, at the start of that set. As mentioned above, this capability is already provided on an optional basis to improve performance in some experimental data base systems. A network model could be implemented by using common field values rather than pointers by "copying" the field value into all members of the set, replacing the pointers. The net result is that either storage structure (pointers or common field values) can support either data model (network or relational). Both of these implementations are in progress in current research activity.

Single data base structure. Another result of the common data base architecture is that a single structure is used for data within a system rather than separate file and data base systems. An example of this is the current use of files for languages such as Cobol. The common data base architecture shows that language "files" can be equally well supported by a data base. If the network data model is used as an example, the file declaration would generate data base subschema, schema, and storage schema. The reads and writes would generate data manipulation commands that would be compiled by the normal data manipulation language compiler. The Cobol program would operate normally and would see no changes in the interface to the data. The data, however, could now be accessed for other purposes through the data base interface. Just as language files can be incorporated into a data base, any file system could be incorporated into a data base by the same general techniques.

It appears that the application of common data base architecture concepts results in data models that are both better suited to end users and simpler to implement and understand. The resulting model should also be more functionally powerful than existing models because it subsumes the functional capability of each of them.

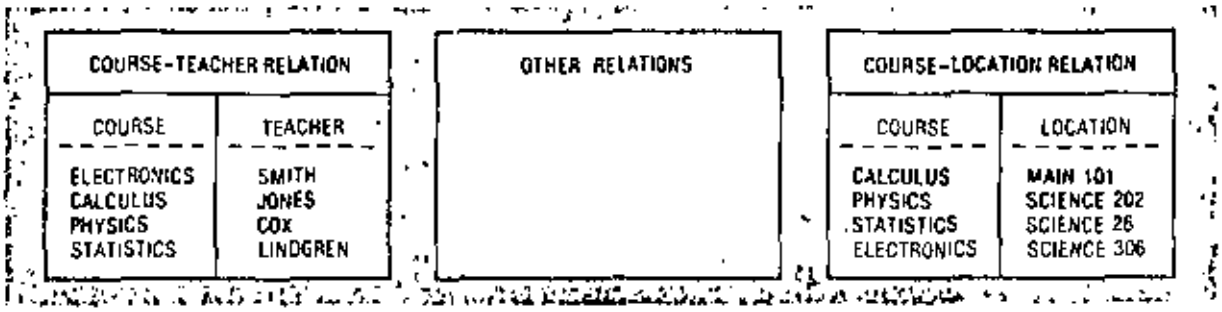


Figure 5. Organization of a relational data base.

A close connection exists between the common data base architecture and data base computer technology. In order to be economically viable, a data base computer must support several data models, specifically including the relational and network models. The common data base architecture provides the underlying mechanism to accomplish this.

Distributed data base systems

It has been suggested that a distributed system is not very useful until the data are distributed. Certainly data play a vital and increasingly important role in modern computer systems. Systems in the past have been implemented using centralized data bases even when other functions have been distributed. The few organizations that have implemented distributed data bases have developed their own systems, because software to implement distributed data bases has not been available from manufacturers.

In the following discussion, the frame of reference is a distributed system that may be worldwide in scope, supporting data at tens or even hundreds of nodes. The nodes are assumed to communicate over heterogeneous communication links that will vary in bandwidth and delay. It is also assumed that not all nodes are able to maintain continuous communications with one another. At the other extreme, and as a subset of the frame of reference, the distributed system may be located within a single building and may require only infrequent interchange of information among nodes.

Objectives and requirements. The objectives of distributing data over the nodes of a distributed computer system are generally the same as of distributed systems as a whole. However, the following objectives are especially relevant to the distribution of data bases:

- performance—to obtain faster response to user queries;
- lowered cost—via reduced utilization, and therefore cost, of data communication;
- shareability—to permit data sharing among geographically separated nodes;
- access transparency—to provide uniform logical access to all locations;
- reliability—to ensure that the system will continue to function adequately in spite of the loss of one or more nodes;
- tunability—to permit data to be distributed to and stored at the location of heaviest usage;
- expandability—to accommodate increases in data base size either on existing nodes or by adding new nodes.

Several of these objectives—notably performance, reliability, tunability, and expandability—can be obtained through the use of redundancy. However, redundancy also introduces the problem of integrity, as we shall see below.

These objectives all represent possible benefits to the user that can be obtained in largest measure through the use of distributed data base systems. Other objectives, which are met by most centralized systems, will still have to be met by distributed systems; they amount to requirements for any integrated, total system:

- recovery—the system must be able to recover automatically following an error, lost message, or solid failure;
- security—users must be able to specify access privileges to secure data that the system administrator can enforce;
- reorganization—it must be possible to reorganize the data to improve efficiency, either locally or globally;
- efficiency—the system must make efficient use of resources through selection of appropriate

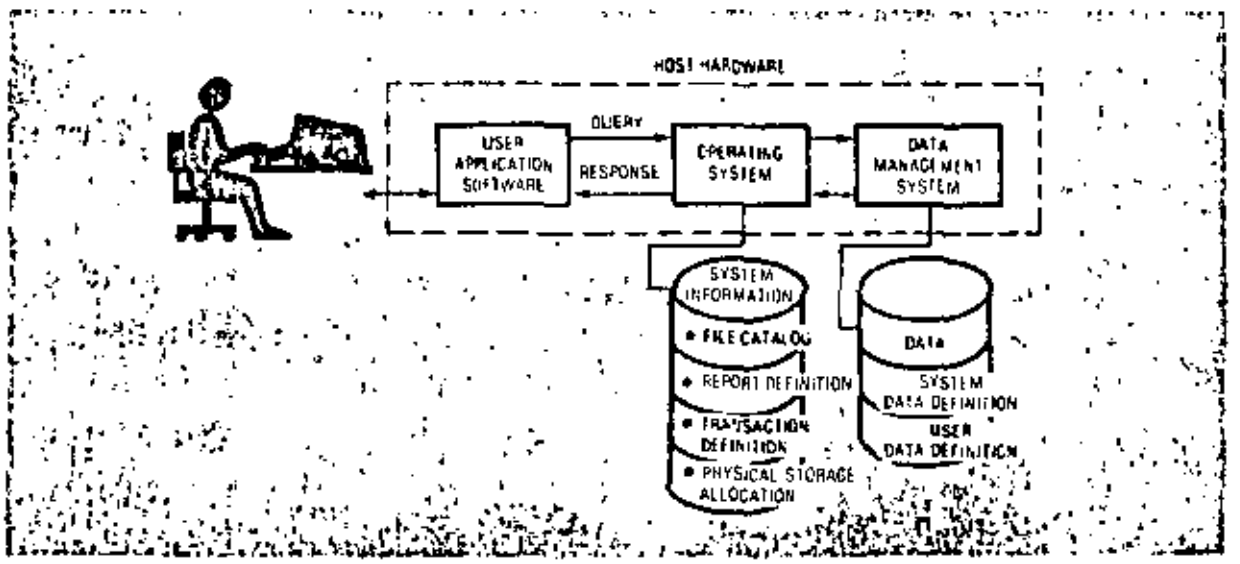


Figure 6. Traditional centralized data base system organization.

algorithms to allow a maximum level of concurrency and a low transaction rejection rate;
- coherency—multiple copies of data must be maintained in a consistent state without this causing unacceptable delays in updating;
- deadlocks—deadlocks must not prevent the completion of transactions;
- fairness—all locations must have equal priority.

These requirements are clearly made more difficult and complex by the distribution of data. Techniques to meet these requirements in a distributed environment are described below. Although these requirements are not in priority order, recovery is probably the most important. Indeed, recovery is the basic design constraint on present centralized data management systems, and will acquire an even greater importance in distributed systems. The other requirements are also important, and any acceptable protocol must be able to handle all of them.

Centralized vs. distributed data base systems. As shown in Figure 6, a traditional centralized data base system—which may be considered to include file systems as a special case where the data definition and access methods are contained implicitly in the user application programs—consists of

- application software,
- an operating system,
- data,
- physical storage allocation,
- data management software,
- data definition (system), and
- data definition (user).

The user and his application program interface to the data management system via the operating system software, which controls all mass storage. If we follow the Codasyl specification terminology, the data are initially loaded into the data base according to the specification contained in the data definition language, which also produces the system copy of the data definition, or schema. The user may also define his own particular definition of the format of that data, called the subschema. The data are manipulated according to the procedures in the data manipulation language, whose statements in the case of the Codasyl specification are embedded in a general-purpose procedural language such as Cobol. The data are stored according to the physical storage allocation determined by the operating system.

In a distributed data base system, these data-related components are placed at the nodes of the system. The question of how to distribute the application software, the data, the data management software, and the data definition immediately arises. Conceivably, each of these could be centralized, or distributed in a number of ways, yielding a very large number of possible combinations. When data are stored at more than one node, there must also exist appropriate data definition and data management software capabilities to access and process the data. When the user request and data exist at different nodes, data management functions must also be distributed.

The integration of data base system technology and a network environment leads to new problems and the need for new functions. A very important need is for a network-wide definition (or directory) of the location and characteristics of all data in the system, including the method of partitioning or replication. Now, when the system receives a user request to access data, it must first determine where the data are located. If the data are not local, the node where the request originates must determine the proper node and communicate with it. If the nodes are not compatible, translation functions must be provided to achieve this compatibility, which may also include request translation, data reformatting, or both.

A distributed data system, shown in Figure 7, contains the same functional components as a central-

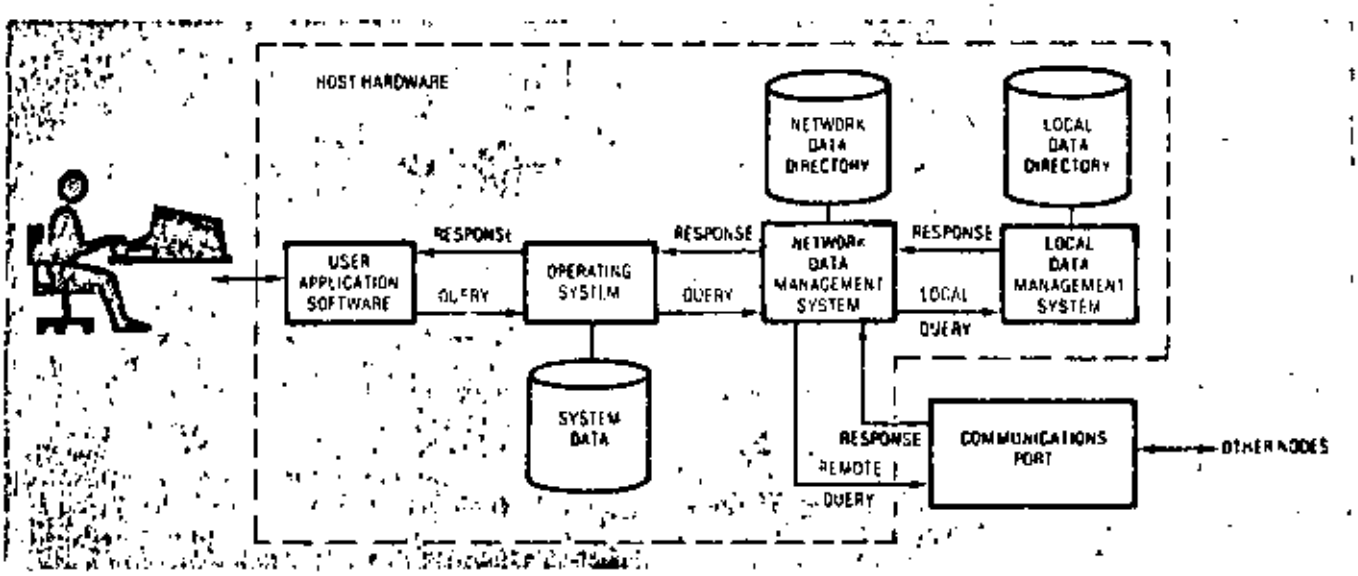


Figure 7. A node within a distributed data base system. Distributed nodes do not have to be "complete" as shown here. For example, some nodes could have data with no local users, while other nodes could have users with no local data.

ized data system, but with additional components: the communications port, the network data directory, and the network data management system. The physical location of the node and the necessary routing information are contained, as usual, in the communications port; the communications port plays exactly the same role in a distributed system that it plays in a centralized system. The network data directory must contain the logical information that relates the various units of data to the node(s) on which it resides. The network data management system must manage all aspects of the geographic distribution of data. These management tasks include

- intercepting the query to determine if it is to local data;
- accessing the network data directory and routing the request to the node holding the data, if the data are nonlocal;
- coordinating all processing and responses, if multiple nodes are involved in an access;
- interfacing with the user, local and remote data management systems, and system directories;
- providing command or data translation in heterogeneous distributed systems.

System structures for distributed data. As mentioned above, a centralized data base system can have a network, hierarchical, or relational data structure. These must be taken as design extremes, because a continuous spectrum of variations exists between these "pure" approaches. In a distributed data base system, although the data are placed at the various nodes within the system, the same structural alternatives are available. It is important to understand that the data structure has no relation to the communication structure. The data structure could be a network (for example, a ring), while the communication structure could be hierarchical; a hierarchical communication structure could support three different data bases—one network, one hierarchical, and one relational.

Other terminology has been used to describe these same basic structures. The "horizontal" data distribution function is the same as a network, because all nodes are at the same functional level. The "vertical" data structure is the same as a hierarchy.

Data distribution alternatives. There are two basic ways in which to distribute data within a system structure: partitioning and replication. When a body of data is *partitioned*, it is divided into disjoint sets, with one set assigned to each node. Only one copy of each record or data item exists, and it is assigned to some node as its home location at any instant of time. Usually, the objective is to assign a record to the node most frequently accessing that record to minimize response time and communications. However, not all accesses at a node (in general) can be satisfied by data at that node, so the system data directory must be used to transfer the access to the proper location. Accesses that cannot be handled locally are called *exceptions*, or *misses*—terms borrowed from cache terminology that are entirely appropriate in this context. There is a strong analogy between cache storage hierarchies and partitioned data base systems. If the application characteristics are such that the data can be partitioned to make all accesses local (i.e., no misses), this represents an extreme case of partitioning, and the data are said to be *segmented* into completely disjoint sets.

When data are *replicated*, two or more copies of the same data exist in the system, and, as an extreme, every node could have its own copy of some data. The benefits of replicating data are that it can provide substantial improvements in performance, cost, tunability, and reliability. However, these benefits come at the cost of greatly complicated update to maintain consistency in the several copies of the data. The topic of updating replicated data has received considerable attention in the research community because of the benefits of replication and is treated in more detail below. Performance is a critical issue in the use of replicated data because several of the algorithms used to maintain data consistency require extensive communication among nodes to avoid long delays in update.

Achieving optimum performance in a distributed data base system is a complex issue; performance depends on miss rates, update frequency, number of nodes, amount of data, and communication speed and cost. Some general guidelines are summarized in Table 1.

Table 1.
Recommended data distribution methods for various combinations of amount of data, update frequency, and miss rate.

AMOUNT OF DATA	REAL-TIME UPDATE FREQUENCY	MISS RATE	RECOMMENDED APPROACH
SMALL	LOW	LOW	REPLICATE
SMALL	LOW	HIGH	REPLICATE
SMALL	HIGH	LOW	PARTITION
SMALL	HIGH	HIGH	CENTRALIZE
LARGE	LOW	LOW	PARTITION
LARGE	LOW	HIGH	PARTITION
LARGE	HIGH	LOW	PARTITION
LARGE	HIGH	HIGH	CENTRALIZE

The most appropriate method of distributing small amounts of data is to simply replicate them if the real-time update frequency is low, because the additional storage cost is low. When the (infrequent) updates occur, they are sent to all locations holding a copy, using one of the synchronization algorithms mentioned later. This approach is especially attractive when updates are batched and high performance and reliability are important.

If the amount of data is small and the update frequency is high, replication becomes very unattractive because of the heavy communications burden. If the miss rate (nonlocal access rate) is low for a partitioned approach (miss rate applies only to partitioned data because replicated data always has a zero miss rate), the best approach is to partition; even though the update rate is high, most updates are to the local

partition. Also, updates to remote partitions require much less overhead and communication than updates to remote replicated data, since only one copy exists. If both the miss rate and update frequency are high, the best approach is to centralize, regardless of the amount of data. (The definition of "high" and "low" miss rates must be made in the context of the communications cost and delay involved. Generally, a miss rate above 30 percent or real-time update rate above 10 percent would be considerable.) If the amount of data is large, the best approach is to partition unless, again, both a high miss rate and update frequency mandate centralization.

Partitioning and replication represent "pure" or extreme approaches. In practice, a user wants to use an arbitrary mixture of both approaches, sometimes for the same data. For example, a data base might be replicated in its entirety at two nodes and partitioned over all other nodes to obtain the tunability characteristic. In other situations, a data base might be mostly partitioned, but selective portions might be replicated at selected nodes to obtain benefits of performance and reliability. The design must be based on patterns of usage or mass storage capability at a specific node (some nodes may not have mass storage). This means that there may be more than one way to reconstruct a complete and nonredundant copy of the logical data from the collection of stored physical data. A collection of physical data that forms a complete and nonredundant copy of the logical data is called a *materialization*.

A complicating factor in partitioning data is the requirement to change the home location of a particular element of data because of changing patterns of access. In principle, this can be done either manually, via the data administrator, or automatically by the system on the basis of instrumentation data. What method is used depends on the dynamics of the application and the courage of the implementers.

In all situations, the users must be shielded from the physical distribution and redundancy of the data and must be able to interact with the distributed system as easily as with a conventional centralized one. The goal of providing a logically centralized interface to users of distributed systems has been addressed in a number of operating systems now in development or in use. Specifically, these included activity by the National Software Works,[22] the National Bureau of Standards Network Access Machine,[10] and the Arpanet RSEXEC project.[24,25] These projects have in common the objective of providing to users a geographicaly distributed computing network resource that is consistent, logically centralized, and easy to use, where access is independent of location, although not always independent of system.

Directories. Just as data can be distributed throughout a system, the system data directory necessary for locating the data can also be distributed. Just as data can be partitioned, replicated, or made redundant in some combination of these, so can the directory. In addition, the directory may be centralized. How the directory is distributed must depend on the same considerations as how the data are distributed, although there is no reason to assume that for a given system the application characteristics of the directory will be similar to those of the data it points to. The data in one system might have a high update frequency, but the directory might have a low update frequency. In another system, the opposite might be true—for example, a system with partitioned data and dynamic (and frequent) relocation of data based on usage. With two "pure" methods of data distribution (partitioned and replicated) and three "pure" methods of directory distribution (partitioned, replicated, and centralized) a total of six pure combinations exist. Ideally, a general-purpose distributed system would support all six pure combinations as well as various mixtures, with the final selection made at system generation time.

The directory itself can be treated as ordinary data, with one critical difference: the location information for the directory must be known to the system nodes a priori. This can be accomplished by storing copies of the directory at every node, since they are moved seldom or never; certainly they are retrieval-intensive.

Treating the directory as ordinary data has another advantage: the normal system functions, such as storage management, recovery, security, and update synchronization, automatically become available for the directories.

Update synchronization of multiple data copies. Because of the several advantages inherent in replicated or redundant partitioned data base systems, considerable study has been devoted within the industry to solving the very difficult problem of update synchronization to maintain data base integrity.

Many applications can function quite adequately with batch updates. However, a number of potential problems exist in *real-time* updates of data with multiple copies. Each node becomes aware of updates originating elsewhere only after some unknown and unpredictable delay. If not controlled, this delay can cause the internal consistency (also called integrity) of the data to be destroyed. For example, if multiple copies of an airline reservation data base existed, terminals attached to two different nodes could each sell the last seat on flight XYZ, then reject the update coming from the other location as causing an oversold condition.

In centralized or nonredundant data base systems, mechanisms such as locks can be set to ensure the serialization of updates.[9,12] Serialization of updates ensures consistency of the data. To maintain consistency of data where multiple copies exist, it is necessary to serialize the updates and to apply the serialized updates to all copies in the same order.

Two levels of consistency can be defined for multiple copies of data. *Strong consistency* is defined as the condition of having all copies of the data updated at the same time. Strong consistency is very desirable, because all copies of the data have the same update status at any time, but this always entails a considerable delay in response time to process the update for all copies. *Weak consistency* is defined as the condition of having the various copies of the data converge to the same update status over time, but at any

instant of time some copies may lag others in the number of updates processed. Delays in general will be reduced, and more efficient use will be made of resources, but some copies of data will be more up to date than others.

We may further define *coherence* as a measure of the differences among multiple copies of data at any instant of time. We also define the *promptness* of a system as the average time delay in completing updates. Formal definitions for metrics of coherence and promptness have been defined.[10] A distributed data base system is *convergent* if the coherence approaches unity over time if update activity ceases.

A distributed data base system cannot be highly coherent and prompt at the same time — perhaps the information-system analog of the uncertainty principle.

A distributed data base system cannot have high coherence and low promptness at the same time; this is a fundamental tradeoff that must be made. This seems to be the information analog of the Heisenberg uncertainty principle in physics, which states that the position and velocity of an object cannot be known simultaneously with arbitrarily high precision.

Some applications require only weak consistency, while others require strong consistency; both types may exist in the same system and possibly require access to the same data, thus leading to some interesting system design problems.

A number of algorithms have been proposed for controlling concurrent updates to maintain consistency under a condition of weak consistency.[10] In the simplest of these, all update requests are sent from their node of origin to a master node, where they are serialized and given a sequence number. The updates are then sent to all centers, where they are implemented in order by sequence number.[11] Other update control techniques are based on a time stamp applied by a clock at the node of origin at the time of receipt. The time stamps can be made unique at the global level if any single node can create at most one update per change of the clock and if the node number is appended to the time stamp. These unique time stamps can be used to serialize the order of updates. One such method is based on a "majority consensus" value.[20] Each update is voted on for acceptance by all nodes based on updates each node has processed earlier. Updates approved by a majority of nodes are accepted in order of their time stamps. This approach is immune to failures in communications links or nodes.

Another approach using time stamps classifies updates according to the variables they modify.[20] A specified node is given the responsibility, a priori, for sequencing updates with variables of a given class. Potential interference between updates of different classes is resolved by a "conflict graph" created by preanalysis of the system. This approach quickly identifies those situations in which no control is required and reserves careful coordination for those updates that could potentially impair integrity.

All of these methods, and others now under study, preserve the integrity of the data. For any given application, there are other goals of importance, including:

- minimizing delay;
- avoiding preferential treatment of nodes;
- minimizing the need to back out updates; and
- minimizing communications cost.

Each update algorithm must be evaluated for the application of interest to optimize these parameters, appropriately weighted, to yield a figure of merit.

Deadlock, the same kinds of deadlock problems that can arise in centralized multiprogramming systems can also arise in distributed systems. The general methods of solution remain the same on a logical level, but are complicated on a physical basis because of the distribution of physical and data resources.

In any approach to deadlock resolution, three basic techniques can be used whether the system is centralized or distributed: (1) prevention, (2) avoidance, and (3) detection and resolution.

Prevention. In prevention, all resources required by a transaction must be requested at the beginning of the transaction. This is very difficult because the resource requirements are often data dependent and not known at the beginning of the transaction. To acquire all possible resources at the beginning would be very inefficient, both holding resources idle and decreasing system concurrency.

Avoidance. Deadlock avoidance requires some advance knowledge of the resource usage of transactions in order to determine, at each point in time, whether the sequences of actions of transactions that have been initiated but not completed are sufficiently valid that the transactions can run to completion. This approach, like the prevention approach, is very unattractive in distributed systems because the necessary advance information to avoid deadlocks is either not available or is distributed so widely as to cause considerable overhead and delay in any attempt to avoid deadlocks.

Detection and resolution. Deadlocks can be detected by means of a search for cycles in a state graph of resource usage. They have been reduced in practice in centralized systems. In a distributed data system, it is generally not efficient to maintain global state graphs for the whole system, as is done for centralized systems. Two methods not requiring global state graphs have been developed for detecting and resolving deadlocks in distributed data base systems; one is hierarchical and one is distributed. Both can be shown to detect all deadlocks.

Performance. Unless careful attention is paid to functions requested and algorithms used, performance can be a problem in distributed systems. An ex-

ample of this is a query that requires cross-referencing of data at several nodes—for example, a join. A task like this can incur significant delay because of the need to bring together at one node all the data required for the execution of the cross-referencing operation.

One approach to implementing operations of this nature is to decompose the query into a sequence of local queries involving only local data, with intermode data transfers between local queries.[7] The algorithm consists of a series of steps, each of which is either a data move or local processing of data that have been moved to a single site. An important assumption of this method is that communications cost will dominate the processing of complex queries in a distributed environment. Therefore, at each step the optimization procedure first attempts to minimize communications cost and then, within the optimal communications strategy, attempts to minimize local processing cost. This will normally result in doing as much local processing as possible to minimize the amount of data communicated among nodes.

Recovery in partitioned and replicated distributed DB systems. In centralized data base systems, recovery from a system malfunction is accomplished by using a combination of (1) before-and-after looks put on an audit trail and (2) checkpoint dumps to establish periodic consistent snapshots of the data. Recovery in a centralized system can be complex and may require manual intervention; recovery in a distributed system can be much more complex, but can also be far more effective if redundant data are used.

Partitioned systems. In a partitioned data base system with no redundancy, the recovery problem for node failures is logically identical to that in a centralized system, and the recovery techniques are quite similar. The only difference is that not all requests are local, but this is only a physical, not a logical, difference. Audit trails and checkpoint dumps continue to be used. When a failure occurs, all transactions are suspended while the system tries to recover using the before-and-after looks. Failing this, the audit trail is backed up to the last checkpoint dump, and the dump is copied into the data base and processed forward, using the audit trail, to the point of suspended operation. At that point, transactions are again accepted. If fast recovery is important, it may be prudent to maintain an up-to-date duplicate copy of the data; this will speed recovery of mass storage failures (but not processor failures).

For communications failures in partitioned data base systems, the originating nodes will fail to get message acknowledgments. The burden of recovery falls on the originating nodes, and access transactions must be queued up until communication is restored.

Replicated systems. In replicated data base systems, the situation is quite different. If either a communications link or a node fails, the rest of the system can continue normal operation, and only the transactions originating at the failed node or link are stopped. When the node or link becomes operational again, two options are available. If the outage was short, the most efficient recovery procedure is to obtain the lost updates from the audit trail of another node. If the outage was long, the best approach would be to acquire the entire data base from a nearby node, along with a time stamp or ticket number for the last update. This would become, in effect, a (very recent) checkpoint dump and could be processed forward with the audit trail to the present time.

It appears that in a replicated data base system there would be no need for nodes to take checkpoint dumps because copies at other nodes could serve that purpose. An exception to this would be if the data base were very large compared to the spare communications bandwidth available, leading to long transmission times. Each node would require an audit trail, just as do nonredundant data base systems.

Lelann[15] describes several protocols for accomplishing recovery in redundant data base systems.

Data base processor architecture

Shortly after the invention of the stored program electronic digital computer in 1946, storage and retrieval of nonnumeric information became an important application. With only a few exceptions, the early file access systems starting in the middle 1950's and the data base systems starting in the late 1960's were mapped onto a conventional von Neumann computer. Although the desirable way to access nonnumeric data is by value, the von Neumann architecture precludes this. Therefore, a number of artificial methods are used to convert a value into an address. These artificial methods include sequential, indexed, hashed, and set access methods. In spite of their indirection, these access methods have successfully met industry needs until the present time. There has, however, been constant research into many aspects of file and data base systems in the general areas of improved functionality, improved performance, and improved availability.

At least one early exception to the use of von Neumann architecture to retrieve nonnumeric data existed; this was the Univac File Computer.* This system, first delivered in 1954, allowed the addressing of data in mass storage by value rather than by address. This was done by storing the value of the desired key in a search register, and then comparing this value sequentially to values on a drum. With this capability, records of up to 120 characters could be stored anywhere on the drum and retrieved by value; no access method was needed. The importance of this capability is only now being rediscovered.

The 25-year quest for improved functionality has led to data base systems as we know them today—initially using network structures, and now with growing interest in relational structures. The performance requirement was met by brute-force improvements in hardware speed. The one element that did not change was the architecture.

Associative storage. Fundamental to the concept of a data base computer is the accessing of data on the basis of value rather than position. Devices that are able to do this have been called "associative stores," "content-addressed memories," or "search memories." The term "associative store" has been applied to two quite different kinds of storage units, with a rather unfortunate confusion of terminology. The first storage units with this capability performed a parallel search in a solid-state storage unit of a few thousand bytes in a time on the order of microseconds. The term used here for this technology is "parallel associative storage"; it is typified today by Staran.[3] The only other application of parallel associative storage devices today appears to be the cache, where the terminology "content addressable memory" is often used. Here the association is within sets of segments of storage, with each set typically having on the order of 1000 elements.

The second kind of associative store uses a sequential storage unit, which may be disk, bubble, or charge-coupled device. This sequential storage unit is searched serially to find data meeting the search criteria. The term used here for this technology is "serial associative storage." Typically, it can search megabytes in tens of milliseconds. Whereas the parallel associative store is limited to equality searches, sequential storage can use greater-than, less-than, and arithmetic and logical expressions involving operands. One of the earliest sequential associative storage devices was used on the CASSM project.[4]

Data base computer architecture alternatives. Beginning in the early 1970's, new architectures were proposed to improve the effectiveness of data base systems. These may be classified as

- back-end systems using conventional minicomputers,
- storage hierarchies (self managed),
- intelligent controllers, and
- data base computers.

These are listed in their approximate order of introduction. I have recently discussed the strengths and weaknesses of these approaches.

Perhaps the earliest new architecture was the back-end processor implemented on a conventional minicomputer.[5] The objective here was both to improve performance and to provide a shared data base to several noncompatible systems. Subsequently, a number of back-end processors implemented on conventional minicomputers have been developed as commercial products.

The storage hierarchy described by Welch in this issue (p. 19) is as old as computers, but the self-managed mass storage hierarchy first implemented in the mid 1970's was a specific attempt to solve file and data base problems. The first implementation was as "virtual disk," where a pool of disks acted as a cache buffer for a tape-cartridge device. A more recent application is the cache disk, where a charge-coupled storage unit acts as a cache buffer for disk (and possibly cartridge) storage. The storage-hierarchy approach depends on substantial locality of reference for data, meaning that if a data item is referenced once, it or a near neighbor is more likely to be referenced again in the near future. A number of experiments performed on a cache disk simulator for a variety of customer files has shown an average hit rate of 75 percent for a four-megabyte cache buffer. This hit rate is sufficient to cut the average access time in half.

Beginning in the mid 1970's, new architectures were considered that were designed to substantially improve efficiency in accessing a file or data base system by using content addressing—that is, addressing by key value rather than by location. All of the approaches were in the form of "back-end" attachments to a conventional host over I/O channels or communications lines, and all are considered intelligent-control-unit approaches. These architectures are, at least implicitly, relational in nature because they use flat file storage structures and because they address information by content. An early example of this kind of architecture is the Content Addressed File System, called CAFS. CAFS is an extension of RARES and is based on fixed-head-disk storage, with on-the-fly processing between read and write heads. A single pipeline instruction stream is used. The CAFS work was among the earliest in content addressing, and CAFS can be considered the baseline system. It had no significant amount of buffering and no parallel processing.

The Relational Associative Processor[6] uses an array of solid-state processing elements, each working on a partition of the total data base. In RAP, parallel processing was introduced, and with it the partitioning of the data base into local storage units.

A rather different approach to a data base processor used a Staran computer.[7] Here, data in a conventional storage unit are sent to an associative register array to be searched for specific key values. The Staran approach is conceptually similar to CAFS, with the fixed-head disk replaced by a random access main store and the simple comparison logic replaced by a large parallel associative store array. An important difference is that Staran can perform multiple-thread searches against a simple file in one pass.

These three approaches, and several other similar ones, greatly improve data base performance, but they suffer from the following drawbacks:

- mass storage is expensive and therefore limited in size;
- the entire data base must be resident in the mass store;
- the full relational join can not be performed;
- sorting can not be performed.

The next step in the evolution of data base architecture was to move from a data base processor to a data base computer.[14] The mass storage medium is moving-head disk, which is an order of magnitude lower in cost per stored bit than fixed-head disk. The concept of partitioning the data base into cylinders is introduced, and hashing of the (possibly several) access fields is used to select the cylinders to be

searched. The information from a disk track is fully buffered rather than processed on the fly. Parallel processing, in a manner similar to RAP, is used to process data from several tracks at once.

More recently, the data base computer concept of Hsiao has been further extended. The objective of this design remains the same as its predecessor: to support network, relational, or hierarchical data bases on the order of 100 megabytes to 100 gigabytes. The design approach also remains the same. It uses a back-end computer with a general-purpose processor controlling an array of processing elements: an intelligent, parallel-transfer disk controller and a parallel-transfer disk. A limited amount of interprocessor communication has been added to permit the join operation and sorting. This communication requires that adjacent processors be able to access the same buffer storage module. A low-cost sequential solid-state associative store has been added to identify unique key or sort field values. The resulting data base computer architecture (Figure 8) is characterized by six major elements:

- a data base processor controller,
- processor elements,
- track buffers,
- a content-addressable memory,
- a parallel-transfer disk, and
- a parallel-transfer disk controller.

The host interfaces to the data base computer using high-level commands, for example of a network or relational nature. These commands are sent to the data base processor controller for analysis. On the basis of this analysis, an appropriate set of parameters is generated and down-loaded to the processing elements. The data base processor controller also generates the necessary I/O commands for the parallel-transfer disk.

The parallel-transfer disk allows the reading of up to all tracks in a cylinder simultaneously; each track of information is transferred to the associated track buffer. Each track buffer serves two adjacent processors. The track buffers are sized to hold four tracks of information each. When the track buffers are loaded, the processing elements begin operating on the data asynchronously, performing selection, projection, joins, or sorts. Data meeting the criteria are passed back to the data base processor controller and then to the host. Relational data base systems can be supported in an obvious manner, and network data base systems can also be supported.

These units operate together synergistically. On search, the search parameters are loaded in to the processing elements so that all tracks in a cylinder can be searched in parallel. The size of the track buffers is adequate to provide for double buffering. The processing elements control the operation of their track buffers. They are sufficiently powerful not only to provide equality matches against parameters in the query, as previous machines do, but also to perform the functions of >, <, ≠, AND, OR, and NOT. Some or all operands can be taken from the record itself ("list all employees where job-title = salesman and bonus > salary").

Somewhat surprisingly, the new solid-state direct access technologies of bubbles and charge-coupled storage do not meet the needs of the track buffer because they are too slow by one or two orders of magnitude. A recent survey has shown that a typical record size is about 200 bytes in length. The most common operation in search is to look at the first few bytes of the key field to see whether or not a record is of interest. If it is not, the next operation is to skip 200 bytes to the next record. RAM can do this in a time near one microsecond, but a charge-coupled store requires 40 microseconds even at a 5-megacycle shift rate. Bubbles are ten times slower than charge-coupled devices and are even less competitive.

Applications of data base computers. Although data base computers substantially improve the efficiency of accessing and processing data base information, it is important to realize that they are not universally better than present techniques. Two conditions are necessary to make data base computers more effective than present architectures:

(1) The data base application must be a sizable fraction of the total system load.
(2) The data access must involve use of several mass storage accesses with conventional techniques.

If the data base load on the system is only 10 percent of the total load, there is very little point in off-loading it to a data base computer. It is only when the load is appreciable—at least 50 percent and preferably 75 percent—that off-loading becomes attractive. Likewise, if the query can be satisfied by a single mass storage access, the data base computer cannot improve efficiency. It is only in multi-access situations that the benefits of the data base computer can off-load the host.

A number of experiments and performance estimates have shown that the data base computer

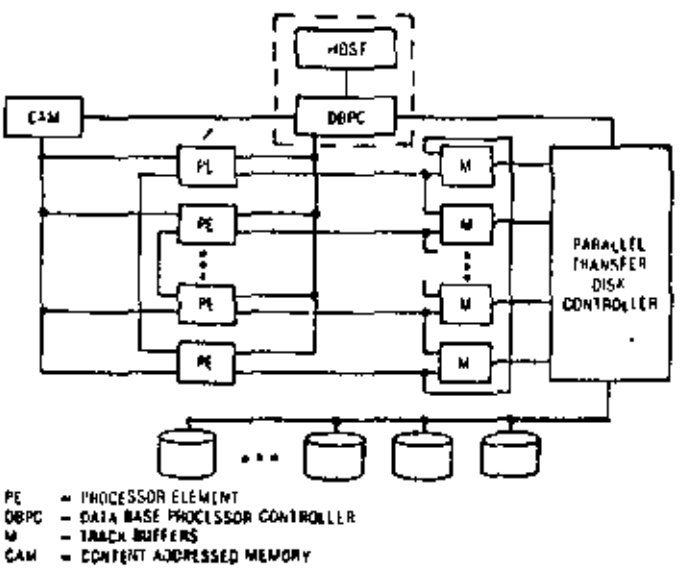


Figure 8. Data base computer organization.

can provide performance improvements of the order 5 to 50 times where it meets the above criteria; in those areas not meeting these criteria, it simply gets in the way and hurts efficiency.

Conclusion

Significant progress in data base technology continues to be made at the logical, physical, and architectural levels, spurred by the combination of user needs and advancing technology. The cost/performance of physical devices is advancing at the rate of a factor of two every two to three years. Significant progress is also being made at the logical level, with the development of a common data model that provides for the translation of schemas between network and relational systems. The common data model also demonstrates that either a pointer-based or a value-based storage structure can be used to support either a network or a relational user interface, thus enabling the support of multiple views of a single data base storage structure.

Building on the advances at the logical and physical levels, the improvements in architecture embodied in data base computer design promise significant increases in performance at relatively modest cost. Thus, significant improvements in cost, performance, and user interfaces are all being made. The future for generalized data base systems looks bright indeed, and we should finally see the removal of the need for each user to "build his own." ■

References

1. Alsberg, P. A. and J. D. Day, "A Principle for Resilient Sharing of Distributed Resources," *Proc 2nd Int'l Conf Software Engineering*, San Francisco, Oct. 1976, pp. 562-570.
2. Astrahan, M. M., et al., "System R: A Relational Data Base Management System," *Computer*, Vol. 12, No. 5, May 1979, pp. 42-48.
3. Berra, P. B. and A. K. Singhania, "A Multiple Associative Memory Organization for Pipelining a Directory to a Very Large Data Base," *Digest of Papers*, COMPCON 76 Spring, San Francisco, Feb. 1976, pp. 109-112.
4. Bobeck, A. H., P. I. Bonyhard, and J. E. Geusic, "Magnetic Bubbles—An Emerging New Memory Technology," *Proc. IEEE*, Vol. 63, No. 8, Aug. 1975, pp. 1176-95.
5. Booth, G. M., "Distributed Information Systems," *AFIPS Conf. Proc.*, Vol. 45, 1976 NCC, pp. 789-796.
6. Canaday, R. H., et al., "A Back-end Computer for Data Base Management," *CACM*, Vol. 17, No. 10, Oct. 1974, pp. 575-582.
7. Champine, G. A., "Four Approaches to a Data Base Computer," *Datamation*, Vol. 24, No. 13, Dec. 1978, pp. 100-106.
8. Copeland, G. P., G. J. Lipovski, and S. Y. W. Su, "The Architecture of CASSM: A Cellular System for Non-numeric Processing," *Proc. 1st Annual Symposium Computer Architecture*, Dec. 1973, pp. 121-128.
9. Eswaran, K. P., J. N. Gray, R. A. Lorie, and I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," *CACM*, Vol. 19, No. 11, Nov. 1976, pp. 624-633.
10. Gelenbe, E. and J. Sevcik, "Analysis of Update Synchronization for Multiple Copy Data Bases," *Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks*, Aug. 1978, pp. 69-90.
11. Grapa, E. and G. G. Belford, *Techniques for Update Synchronization in Distributed Data Bases*, Center for Advanced Computation Report, Univ. of Illinois, 1977.
12. Gray, J. N., R. A. Lorie, G. R. Putzolu, and I. L. Traiger, *Granularity of Locks and Degrees of Consistency in a Shared Database*, IBM San Jose Laboratory Report, 1975.
13. Hoagland, A. S., "Storage Technology: Capabilities and Limitations," *Computer*, Vol. 12, No. 5, May 1979, pp. 12-18.
14. Hsiao, D. K. and S. E. Madnick, "Database Machine Architecture in the Context of Information Technology Evolution," *Proc. Third Int'l Conf. Very Large Data Bases*, Tokyo, Oct. 1977, pp. 63-84.
15. Johnson, H. R. and J. A. Larson, "A Common Data Base Architecture," Internal memo, Sperry Univac Corp., Aug. 1978.
16. Lelann, G., "Algorithms for Distributed Data Sharing Systems Which Use Tickets," *Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks*, Aug. 1978, pp. 259-272.

17. Lin, S. C., D. C. P. Smith, and J. M. Smith, "The Design of a Rotating Associative Memory for Relational Database Applications," *ACM Trans. Database Systems*, Vol. 1, No. 1, Mar. 1976, pp. 53-75.

18. Panigrahi, G., "Charge-Coupled Memories for Computer Systems," *Computer*, Vol. 9, No. 4, Apr. 1976, pp. 33-42.

19. Rosenthal, R., *A Review of Network Access Techniques with a Case Study: The Networks Access Machine*, NHS Technical Note 917, July 1976.

20. Rothnie, J. B., N. Goodman, P. A. Bernstein, and C. A. Papadimitriou, *The Redundant Update Methodology of SSD-1: A System for Distributed Databases*, Technical Report No. CCA-77-02, Computer Corporation of America, Feb. 1977.

21. Rothnie, J. B. and N. Goodman, *A Study of Updating in a Redundant Distributed Database Environment*, Technical Report No. CCA-77-01, Computer Corporation of America, Feb. 1977.

22. Schantz, R. E., and R. E. Millstein, *The FOREMAN: Providing the Program Execution Environment for the National Software Works*, Report No. 3266, Bolt Beranek and Newman, Inc., Mar. 1976.

23. Ozkarahan, E. A., S. A. Schuster, and K. C. Smith, "RAP—An Associative Processor for Data Base Management," *AFIPS Conf. Proc.*, Vol. 44, 1975 NCC, pp. 379-387.

24. Thomas, R. H., "A Resource Sharing Executive for the Arpanet," *AFIPS Conf. Proc.*, Vol. 42, 1973 NCC, pp. 155-163.

25. Thomas, R. H., *A Solution to the Update Problem for Multiple Copy Data Bases Which Use Distributed Control*, Report No. 3340, Bolt Beranek and Newman, Inc., July 1975.

26. Thomas, R. H., *A Majority Consensus Approach to Concurrency Control for Multiple Data Bases*, Report No. 3733, Bolt Beranek and Newman, Inc., Dec. 1977.

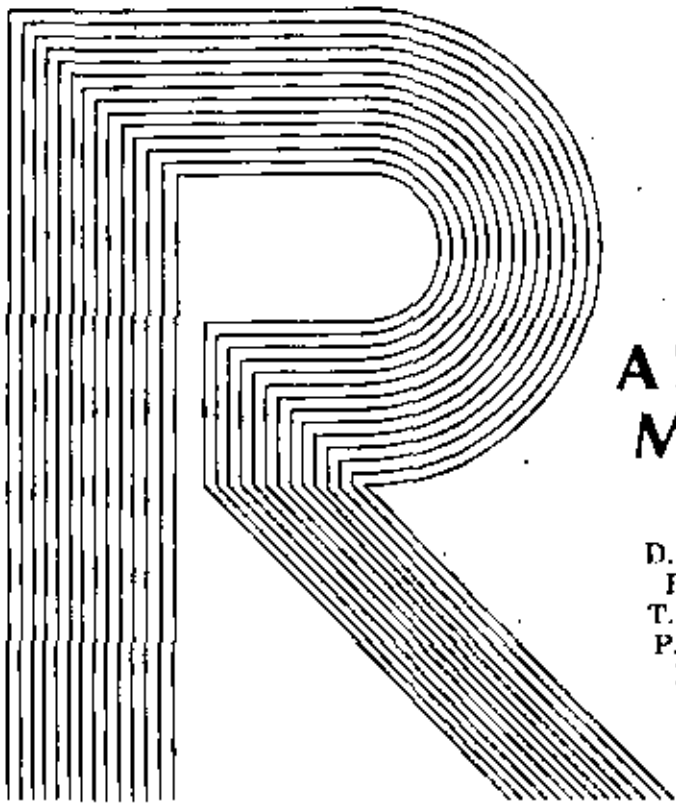
27. Welch, T., "Memory Hierarchy Configuration Analysis," *Computer*, Vol. 12, No. 5, May 1979, pp. 19-26.

George A. Champine is director of advanced systems for large-scale commercial computer systems at Sperry Univac. He is responsible for the technical planning, design, and analysis of systems beyond those currently committed to product. He is also responsible for design and management of large special projects using commercial equipment. In his 20 years with Sperry Univac, he has held several technical and managerial positions in the software and system design fields. His most recent prior position was senior staff consultant, where he was responsible for the management of the advanced technology program for large-scale computer systems.

Champine is a visiting associate professor at the University of Minnesota, where he teaches courses in advanced information systems design and analysis, and engineering project management. He has authored a number of technical papers, made presentations at national computer conferences, and published in the computer trade press. He has made a large number of presentations on computer technology and economics in North America, Western Europe, Australia, Japan, and South Africa.

Champine holds BS and MS degrees in physics, and a PhD in management information sciences, from the University of Minnesota.

A relational approach makes this experimental data base management system unusually easy to install and use. Some of the decisions made in System R design in order to enhance usability also offer major bonuses in other areas.



System R: A Relational Data Base Management System

M. M. Astrahan, M. W. Blasgen,
D. D. Chamberlin, J. N. Gray, W. F. King,
B. G. Lindsay, R. A. Lorie, J. W. Mehl,
T. G. Price, G. R. Putzolu, M. Schkolnick,
P. P. Selinger, D. R. Slutz, H. R. Strong,
P. Tiberio, I. L. Traiger, B. W. Wade,
R. A. Yost

IBM San Jose Research Laboratory

Perhaps the greatest impediments to the use of a computerized data base management system are installation cost and complexity. At present, installation of these systems requires a staff skilled in telecommunications, operating systems, data management and in applications. In response, our lab designed and implemented System R—an experimental data base management system allowing easy definition of data bases and data base applications without sacrificing the function and performance available in most commercial systems. Among its capabilities, the system provides a sophisticated authorization facility, and automatically handles system functions such as recovery and concurrency control. System R adopts a relational data model and supports a language called SQL, for defining, accessing, and modifying various views of the data base.

The relational data model

All data base management systems represent data in the form of *records*. Records, the basic unit of storage, contain *fields*, which hold values. Each record represents some fact about the world. A telephone record, for example, has three fields: name, address, and telephone number. Each real-world *instance* of the record assigns a value to each field.

Data base management systems differ in the way they organize records. System R organizes data into *tables*—sequences of identically formatted records. Train schedules, price lists, tide tables, and phone books all fit this model.

A relational model was adopted because it is easy to understand and to explain and because it lends itself to powerful *relational operators* (such as *sort*, *join*, *project*, and *select*).[2] By relational model, we mean an organization which collects data into just such uniform tables as those discussed above, and which allows a user to access data without having to specify the physical organization of the tables. Figure 1 shows a fragment of a relational data base; the various attributes—name, office, job, salary—define a *relation*. Each record (row), therefore, is an instance of that relation.

Relations are combined and manipulated by various relational operators. Each operator produces

0018-9162/79/0600-0042$00.75

another relation as a result. Operators apply to entire relations, thereby reducing the use of constructs such as "for each record do." Operators include the following: *select* choses a row subset of a table based on a predicate; *project* removes columns of a relation; *join* combines two or more relations to produce a new relation; *order* sorts a relation by a collection of attributes; *group by* aggregates records by some attribute; *Boolean operators* provide union and intersection of relations; and *aggregate operators* such as *sum*, *min*, and *max* collect all instances of a field into a single value.

By contrast, other systems present a hierarchical or network data model.[4] In general, one must *navigate* through a network or hierarchy. For example, to find recent invoices of a customer, we locate the customer account record and then locate the invoice records *under* that customer. By contrast, a nonnavigational approach requests all account records of that customer and relies on the system to locate them. Navigation is not inherent in nonrelational systems, but to date all nonrelational systems have provided a navigational interface.

The three data models have equal power of expression. Besides being easier for the end user to visualize and understand, the tables of a relational DBMS make it somewhat easier to define a nonnavigational language. Further, it appears possible to compile code from a nonnavigational language which rivals the execution efficiency of navigational interfaces.

For these reasons, System R supports only a relational model at the external interface; that is, the user can work in a set-oriented language without describing the pathways taken to reach the data. Since efficient implementation of the relational model and operators seems to require the use of network mechanisms (pointers among related records), System R has full network support internally. However, all this is hidden by the SQL language.

High-level and host language

A very high-level data access language (such as SQL) for data definition, manipulation, and control is important for several reasons:

- It is easy to learn and use.
- It permits the use of an optimizer to improve performance.
- It provides some independence between the application programs and the stored data, so that if the data is reorganized, the applications do not have to be rewritten.

SQL is an example of such a language. (A complete definition of that language appears in Astrahan, et al.[1]) For instance, in the sample data base of Figure 1, we could find who does what in the Paris office by writing:

```
SELECT    NAME, JOB
FROM      EMPLOYEE
WHERE     OFFICE = 'Paris';
```

EMPLOYEE

NAME	OFFICE	JOB	SALARY
Smith	Paris	SALES	15000
Jones	Bonn	SALES	18000
Clark	Boise	SALES	12000
Jones	Boston	SERVICE	17000
Kent	Paris	SERVICE	15000
Davis	London	SERVICE	13000
Jacob	Rio	SALES	12000
* * *			

OFFICE

LOCATION	MANAGER	PHONE
San Jose	Blasgen	7152
Paris	Portal	9123
London	Portal	3278
Bonn	Roever	1287
* * *		

Figure 1. A fragment of a relational data base. All data is organized into tables, i.e., into sequences of records, each of which has the same format. Each row of a table (e.g., see "Jones") is a record describing one instance of a relation.

In many data base systems, data can only be accessed and manipulated by writing a program in a language such as Cobol or PL/I. Such a program contains CALL statements to subroutines performing data base functions. This has several drawbacks:

- It requires the services of a Cobol or PL/I programmer.
- It introduces a long delay between request and result (write the program, compile it, debug it, etc.)
- Using the CALL statement prohibits any compile-time optimization of the data base services required.

The solution to the first two drawbacks is to introduce a *query* capability, that is, a mechanism supporting *ad hoc* requests to retrieve or modify information in the data base. For example, System R implements this query capability by providing a host-language interface as an application program called the UFI—user-friendly interface. Thus, data stored in a System R data base can be accessed and updated both through the UFI (by interactively issuing SQL statements) and by application programs written in host languages such as Cobol and PL/I (see Figure 2).

The solution to the performance problems caused by the third drawback is to extend the host languages to include the data sublanguage. Cobol and PL/I have been extended to permit SQL statements be imbedded in an application program. The extensions are via preprocessors which accept COBSQL and PLISQL input programs and produce Cobol and PL/I output respectively along with com-

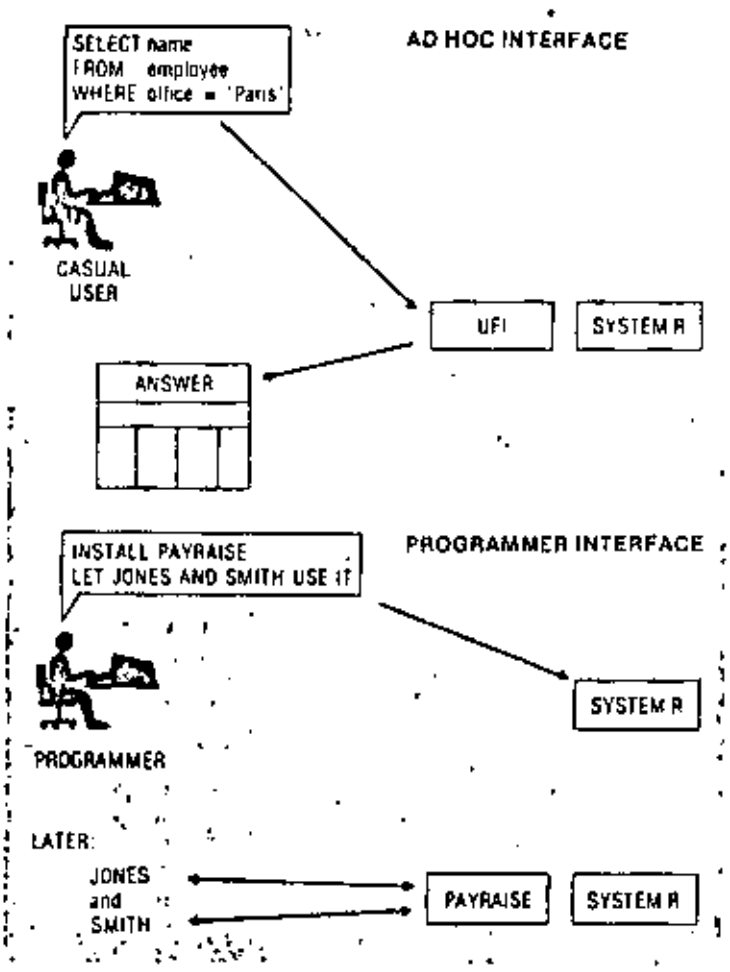


Figure 2. These diagrams show two scenarios, one using the ad hoc interface and the other the programmer interface. In the first scenario, a casual user deals directly with UFI—the User Friendly Interface. In the second scenario, a programmer installs an application program; then Jones and Smith use that application.

piled fragments of SQL statements; these fragments are stored in the data base by the preprocessors. An example application program using PL/ISQL is shown in Figure 3.

The SQL language used in host languages is the same language accepted by the UFI; hence the SQL portion of application programs can be debugged using the UFI. Since the SQL statement is an integral part of the application program (and not a parameter of a CALL), it is possible to perform early binding (compilation) and thus enhance performance.

Compilation

A major criticism of nonprocedural languages is that they are inherently inefficient. If that applied to SQL, System R would be of little interest. Our work, therefore, concentrated on performance as well as on function. To achieve acceptable performance, System R compiles SQL statements into machine code containing calls to low-level accessing routines. The alternative is to interpret each statement every time it is executed. Compiling an SQL statement captures and preserves information that an interpreter must rediscover on each invocation. Experiments indicate compilation is almost always superior to interpretation, even for SQL statements which are executed only once and which retrieve or modify only a few records (see Figures 4 and 5).

In fact, System R can be thought of as a compiler of data manipulation statements. It compiles statements in the SQL language into machine code; the code issues calls to the access method.

Prior to PL/I compilation, a PL/ISQL program (such as in Figure 3) is examined by a System R preprocessor which finds the SQL statements. Each SQL statement is passed to the System R parser, optimizer, and code generator. These produce object code containing calls to a relational access method. The PL/ISQL program is then translated into a pure PL/I program containing calls to this object code. As a result, the cost of supporting the high-level language SQL is paid once, at compile time, rather than at run time. If a query is used many times, its one-time cost is amortized over many invocations.

An optimizer determines which access paths to use to evaluate a SQL statement. It is the optimizer's responsibility to develop an optimal "plan" for the evaluation of each statement. This optimal plan minimizes the execution cost in terms of instructions and I/O operations. The plan is then passed to the code generator, which creates a machine language program to carry it out.

Nonprocedurality and automatic path selection

Nonnavigational data base languages allow data accesses and updates to be expressed without implying either the existence of specific access paths or the physical layout of data. While this makes application programs simpler, it uses the DBMS to choose an optimal strategy for evaluating the program. This system is also relatively flexible. In particular, it is hard to imagine how a system without a high-level language would allow programs to adapt to new storage structures. Using the Figure 1 example, if we wish to determine if manager Portal has a service person in his office, we just query:

```
SELECT   NAME
FROM     EMPLOYEE, OFFICE
WHERE    EMPLOYEE.OFFICE
           =OFFICE.LOCATION
AND      OFFICE.MANAGER='Portal'
AND      EMPLOYEE.JOB='SERVICE';
```

Since the language specifies only *what* is desired, and now *how* to obtain it, the optimizer chooses among several possible plans: One strategy is to search EMPLOYEE looking for service people and, for each service person, use the corresponding OFFICE to enter into the OFFICE table to see if that employee works for Portal. Another strategy might first search OFFICE to find what LOCATIONs Portal manages and then search EMPLOYEE for

service people at those locations. Other strategies involve sorting one or both tables.

When the System R optimizer selects the minimum-cost strategy for carrying out a statement, the cost is based on estimates of CPU and I/O requirements. Using an optimizer in this way has two benefits: First, the user need not be concerned with storage details. Second, the user is prohibited from "taking advantage" of knowledge of such details. The second benefit allows the program to continue functioning as the underlying storage structures evolve with time.

Data independence

Three techniques enhance System R performance: First, the high-level SQL allows global optimization of the query and the consequent generation of efficient code (minimum number of data base calls and I/Os). Second, the system can be tuned by defining or deleting access paths (either associative or pointer-based). Lastly, one can specify that records be clustered together in physical media for fast sequential access. All three techniques are transparent to the application programmer and the decisions can change without altering any programs.

To illustrate the point that System R gives programs some independence from data reorganization as the system is tuned, suppose there are two transactions which must be run periodically against the data base in Figure 1.

(T1) List all employees at a given office.
(T2) List all employees with a certain job.

These two queries need not consider how or where the records are stored nor specify the access paths for locating the records. The transactions will work correctly no matter how the data is organized.

If the users expect to run T1 ninety-nine percent of the time, and T2 one percent of the time, a data base structure which results in high performance will include a fast way to find all employees in a given office, (e.g. an index on the OFFICE field of the EMPLOYEE table), with records of employees at each office clustered together in secondary storage (to minimize I/O).

An index is a means of directly addressing records of a relation containing a common value in a particular column, such as EMPLOYEE, where OFFICE='Paris.' Any authorized user can define the necessary index and clustering criterion and have the data reorganized appropriately. The T1 and T2 transactions will then take advantage of this new organization.

Suppose, however, that these estimates are wrong or that the use of the system changes so that T2 runs 99 percent of the time. Performance on the above data base structure will be very poor, and the data base will have to be restructured. Namely, the OFFICE index will be dropped, and a clustering index on JOB created. An essential characteristic of a *flexible* system is that this restructuring not require that programs T1 and T2 be rewritten.

```
PAYRAISE  PROC (OFFICE);                  /*gives each member of     */
                                          /*some office a 10% raise  */

  $DCL (NAME, OFFICE, SALARY, RAISE);     /*local declares           */

  $LET MEMBERS BE                         /* defines the set of      */
          SELECT NAME, SALARY             /* members to get the      */
          OTID   $NAME, $SALARY           /* raise.                  */
          FROM   EMPLOYEE
          WHERE OFFICE = $OFFICE;

  $OPEN MEMBERS;                          /* start enumerating set   */
  DO WHILE ('1'B);

    $FETCH MEMBERS;                       /* fetch next set member   */
    IF SYR_CODE ¬= 0 THEN                 /* if no such member then  */
      DO;
      $CLOSE MEMBERS;                     /* close set and return    */
      RETURN;
      END;

    RAISE=SALARY*0.1;                     /*else compute 10% raise   */

    $UPDATE  EMPLOYEE                     /*and update his salary    */
       SET   SALARY = SALARY + $RAISE
     WHERE   NAME = $NAME;

  END;                                    /*end of set iteration     */
END;                                      /*end of procedure         */
```

Figure 3. A sample PLISQL program illustrates this high-level set-oriented language accessing and operating on a table, or view of a table, without detailed description of the specifics of the table organization.

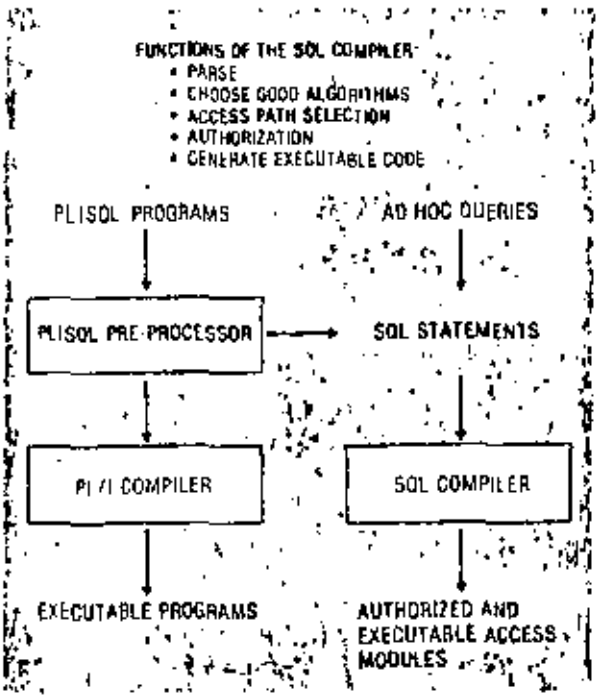


Figure 4. Data flow and summary of SQL compiler. The compiler contains a statement-parser, an optimizer, and a code-generator.

In general, since System R supports a very high-level language, most data base structuring issues can be deferred until after the applications are written. This install-now-tune-later philosophy also eases application programming by deferring many performance decisions, such as the choice of indexes.

Incidentally, System R includes a utility program which evaluates a set of transactions and their relative execution frequencies and suggests good data base structurings for the user. This partially automates the data base design task.

Integrated data dictionary

A data dictionary is a description of the data base. It contains both machine-readable and human-readable descriptions of the data base tables, their attributes, interrelationships, and meaning. It is usually not very large, but it has a very rich structure.

Most systems have a data dictionary facility which stores *metadata* about the data base aside from the data base itself. The data dictionary is often built on top of the DBMS as a special application with a special data definition language. This approach allows the data dictionary to benefit from the facilities of the DBMS.

Separating the data dictionary from the data base raises two problems: First, the dictionary and data base may disagree with one another unless one interface has control of both functions. Second, having a separate data dictionary implies having a separate language for the definition and manipulation of the dictionary data base.

SQL is an integrated data definition and data manipulation language. In System R, the description of the data base is stored in user-visible *system* tables which can be read and altered using the SQL language. The creation of a table or an access path results in new entries in these system tables. Users who define tables and other objects are encouraged to include English text to describe the *meanings* of the objects. Later, other users can retrieve all tables with certain attributes or can browse among the descriptions of defined tables, if they are so authorized. A user can modify these entries to change the attributes of an object. This approach eliminates the dual language problem and assures that the dictionary agrees with the actual system.

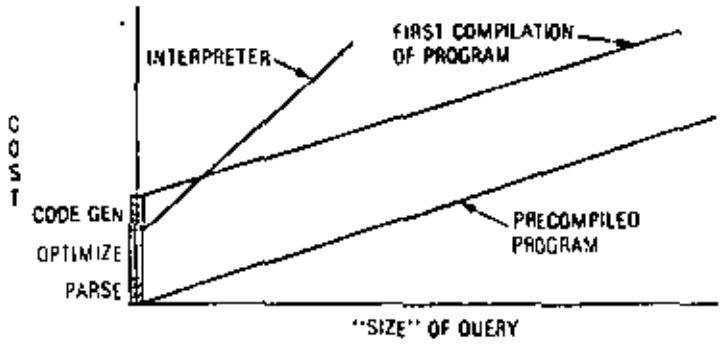


Figure 5. A comparison of interpretation, compilation, and precompilation shows the decided advantage of amortizing costs of screening and precompiling SQL statements, especially when a query will be used many times.

Views and authorization

The result of any SQL query is itself a table. A user can display such a table immediately, or store the definition of the table as a *view*. Views can be used just like other tables, except that some views can not be modified (will not support insert, delete, or update).

Views provide data independence. If the structure of a table is changed (columns added or permuted or a table split into two tables), the user can define a view which looks like the original table. Old programs can access the new data via the view.

Views also provide a powerful authorization mechanism. Rather than allowing access to an entire table, we can define a view which is a row and column subset of the table and only allow access to that view. For example one view might allow a manager to see records in his own department only. Further, certain fields of the view might be available for read only. This gives value-dependent authorization at a very detailed level.

Views, in combination with granting and revocation operations, provide the basis for the authorization mechanism. System R maintains special tables containing the definition of each view and the operations which selected users may perform on that view. In order to allow either centralized or distributed control of access, a special privilege called *grant* is included. Grant allows one user to grant any subset of capabilities to other users, who can pass on the grant as well. The compilation process checks these authorization constraints (at some cost), but this cost is paid only once, at compile time. If the query is invoked many times, a simple test at each invocation ensures that the decision is still valid. This powerful authorization system thus incurs almost no run-time overhead for compiled transactions.

In System R, there is no centralized DBA—data base administrator—function. Rather, each user group can have its own DBA to create and authorize access to that group's data. This provides a common data base shared among all users (an integrated data base) and yet allows some autonomy among the user groups.

Further, authorized users can perform almost all DBA functions at any time without interrupting the normal operation of the system. Authorized users can

- create and destroy tables,
- create and destroy indexes on tables,
- add a column to an existing table,
- install a new transaction,
- add users to the system,
- change the privileges held by various users, and
- define or drop a view of existing data.

Since SQL supports these operations, they may be invoked from a terminal (via UFI) or from a program.

Installation and tuning

The installation procedure for System R consists of acquiring storage space for the code and data of a starter system, which contains a small data base (relating to employees of a hypothetical company). Once this starter system is up and running, it enables users to experiment with a working version of System R in their own environment. When they have familiarized themselves with the starter system, they can begin to define their own data bases and transactions; this can be done in an incremental and experimental manner. Users can quickly define and use new data bases because the system permits them to defer many data base specification problems.

After an application is coded and running, the user can tune the system with a program which, for a set of SQL statements and their probabilities of occurrence, evaluates possible data base specifications and tells the user the best ones. A proposed data base specification consists of a clustering criterion for each table and a set of indices to be maintained on each table. The program also estimates the cost of evaluating each SQL statement and the weighted cost of the application.

Transaction management

A major goal of System R is to provide a full set of capabilities for data base management in a realistic, operational environment. Only in this way can we assess the viability of the architecture. In System R, multiple users can concurrently access data, and the system has complete facilities for transaction backout and system recovery. Recovery compensates for system failures as well as catastrophic failures of the magnetic media (e.g. a disk-head crash). We tried to avoid the need for human intervention in system recovery. In particular, recovery requires no explicit operator commands; only media recovery requires handling magnetic tapes. Almost all recovery information is kept on disk and a noncatastrophic restart does not involve operations personnel.

The transaction concept is the key to a successful recovery philosophy. A complex update of the data base involves many SQL statements. If a complex update fails or if the system crashes during the operation, the state of the data base will probably be confused. The system must be able to *undo* partially completed transactions. System R does this by keeping a log of all the changes a transaction has made. For example, it keeps the old and new value of each updated record. If the transaction gets into trouble, the system can undo the transaction by setting updated records to the old values still in the log. Transactions may be undone, by either a system or an explicit user request. Once a transaction *commits* its updates, they will never be undone. Committed updates are redundantly stored in the log so that they can be reapplied in case of system or media failures (see Figure 6).

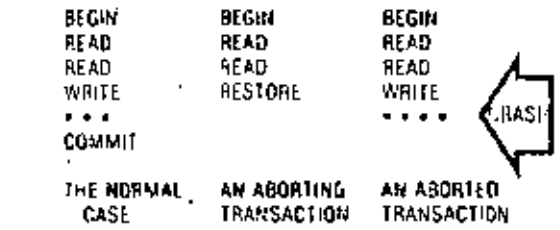


Figure 6. This illustrates the three possible transaction outcomes; the transaction commits, aborts, or is aborted by the system. A transaction is a unit of recovery; it commits all outputs, or aborts and has no effect. A transaction is also a unit of consistency; it starts with consistent inputs and produces consistent outputs.

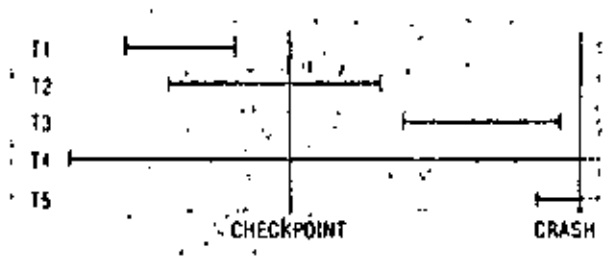


Figure 7. Five transaction types with respect to the most recent checkpoint and the crash point. At restart, T2 and T3 will be redone while T4 and T5 will be undone. Since T1 ended before the checkpoint, its outputs already appear in the checkpoint.

The log is also useful in reconstructing a current version of the data base from an archive or checkpoint version. We use a new technique which provides incremental checkpointing to disk and yet permits dynamic space allocation. Incremental checkpointing allows us to use a particularly simple system restart technique. When a system crash occurs, a previous data base state is rapidly recreated through the use of backup page maps and checkpoint images of modified pages. This data base state, coupled with transaction log information, is then used to either redo or undo all transactions which were in-progress at the checkpoint or completed after the checkpoint (Figure 7).

The transaction concept is exposed to the user by providing operations to begin, undo, and commit the work of a transaction. The application programmer brackets a successful transaction by a BEGIN-COMMIT pair. An aborted transaction ends with a RESTORE (undo) verb. All other aspects of recovery are handled by System R.

Transactions also supply the key to concurrency control. If multiple transactions concurrently read and write the same data, anomalies can occur. For example, if two transactions both want to add $100 to an account having $1000 balance, then the proper outcome should be $1200. But if both transactions

start with the $1000 balance, then the outcome will be $1100. For this reason the execution of the two transactions T1 and T2 is serialized.

System R uses a locking protocol so that: (1) the system itself never gets confused because of concurrent access to a data item by two or more transactions, and (2) the user can control the extent to which his transaction is isolated from the effects of other transactions. Each user can select one of three isolation levels:

- The lowest level permits the reading of data which has been updated by an incomplete transaction.
- The next level guarantees that only committed data is read, and
- The third level guarantees that all reads are both committed and repeatable.

At all levels, System R prevents updates on top of uncommitted updates, so that the system can undo the effects of each transaction independently. All locks are set automatically to ensure the semantics of these isolation levels. The locking subsystem handles queuing and deadlock detection. The transaction recovery mechanism resolves deadlocks by undoing one or more transactions and pre-empting their locks.

Status and experience

SQL is accessible from PL/I and Cobol and runs on IBM 370 VM and MVS operating systems. System R has been operational since mid-1977. Since then, it has undergone experimentation in various environments. Experience thus far indicates that the system is indeed easy to learn, install, and use. Further, its performance, when approached at the SQL level, is comparable to other data base management systems.

Typical System R applications involve business operations such as accounting, inventory control, purchasing, and bill of materials tracking. For the most part, these applications use System R through a host programming language (one would not want to generate the monthly statements of a large corporation by using an ad hoc query terminal).

Users nevertheless make considerable use of System R's query facilities. For example, they can debug SQL statements in advance by testing them using the UFI. Another example involves table loading—to assure that the table was correctly loaded, it is easier to verify the values using the UFI than to write a complex but seldom-used verification program. These examples demonstrate that it is important to offer both query and host language support in a complete data base management system.

Experience also shows the importance of providing frequently used transactions with high performance by means of early binding (compilation). Consider, for example, an airline reservation application. The most common and simplest instance of this application is fetching one record (a flight) and decrementing the number of seats available. Using the access method, the cost for this operation is very low, and one cannot afford to pay the cost of optimizing and interpreting the transaction on each execution.

In general, our experience with System R indicates that the approach will yield good performance for a variety of users, along with the important ability to tune the system as performance requirements change. In addition, the system is easy for new users, who can employ high-level languages to handle both new and existing applications. ■

Acknowledgments

Many have contributed to the design and implementation of System R: Raymond Boyce, Arvola Chan, David Choy, Kapali Eswaran, Ted Haerder, Randy Katz, Won Kim, Leonard Liu, Paul McJones, Moscheh Mresse, Jorgen Nilsson, Scott Parker, Dominic Portal, Phyllis Reisner, Paul Roever, Robert Selinger, and Vera Watson.

References

1. M. M. Astraban, et al., "System R, A Relational Approach to Database Management," *ACM Trans. Database Systems*, Vol. 1, No. 2, June 1976, pp. 97-137.
2. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *CACM*, Vol. 13, No. 6, June 1970, pp. 377-387.
3. Special Issue: Data Base Management Systems, *Computing Surveys*, Vol. 8, No. 1, Mar. 1976.

centro de educación continua
división de estudios de posgrado
facultad de ingeniería unam

CASOS PRACTICOS EN EL DISEÑO DE SISTEMAS DE INFORMACION

EVALUATION CRITERIA FOR LOGICAL DATABASE DESIGN METHODOLOGIES

DR. ALEJANDRO BUCHMANN

AGOSTO, 1980

Evaluation criteria for logical database design methodologies

A P Buchmann* and A G Dale†

The increasing acceptance of database systems has shown the need for logical database design methodologies and evaluation criteria to select the one most appropriate for a particular task. Application of currently existing methodologies to a portion of the data handled during chemical process plant design has highlighted strengths and shortcomings of the methodologies analysed. Using these results, evaluation criteria are established and synthesis of a methodology which is responsive to the identified criteria is proposed.

INTRODUCTION

Interest in database design methodology has developed in the context of applying database management technology to support the integration of engineering design of chemical process plants.

In all engineering disciplines evolution has led from intuitive to systematic design procedures. Database design is no exception. Many investigators in the area have recognized the need for a systematic approach. Their attempts have been reviewed by Fry and Novak[1] and Yao Navathe and Weldon[2]. While existing attempts at proposing a database design methodology have many points of interest, they rely to some extent on shortcuts and intuitive assumptions. A logical database design methodology which does not omit any steps, and which covers the whole range of logical database design activities is required.

In this paper the environment in which the present work evolved is discussed along with a view of the database design process and the reasons for use of database design methodologies. Some of the proposed methodologies and how they perform in the environment analysed are presented. From their observed strengths and shortcomings a set of criteria for evaluating a methodology is derived. The last section suggests the synthesis of a new methodology responsive to the stated criteria.

ENVIRONMENT

Databases can play several basically different roles in supporting the engineering design of a chemical plant. For example:

- as an updatable description of the current status of a project, from which information used as input to a particular task can be obtained, thus enhancing communication amongst users
- as a private working-space for a small group of users serving as an archival system for preliminary design alternatives.

If the database is viewed as a record of progress of a project, it will serve as a communication medium among groups of designers carrying out various tasks in the overall design and will contain already-approved data released for use by other groups. Well-defined channels of communication exist for this purpose within an organization and information is traditionally presented to the user in form of familiar documents, e.g. flow sheets, load sheets, schematic P and I diagrams, specification sheets, etc. Data have to be supplied also to design software packages which should rely primarily on the information contained in the project-wide database. This type of database reflects the gradual refinement of the project and grows monotonically. Various degrees of detail have to be represented, depending on the stage of completion of the project. For example, bidding information can range from coarse estimates to data obtained from an almost completed design, but will be in most cases less detailed than information used for subcontracting or the shop information and the final project documentation. The dependence of many applications on the same data or dependence on data generated by intermediate applications cause any update to propagate throughout the database. An example is the process design stage whose output serves as problem definition for later tasks in the design chain. Updates, therefore, have to be well-documented in the form of revisions and alerts to users of the updated information are necessary.

In particular, the interfaces among the process engineering group, the systems group and one equipment design group, the heat exchanger design group are examined. Information is transferred from the process engineering group in the form of a process release, consisting of a flowsheet, a material and energy balance, a major equipment list with identification of accounts, and load sheets. Detailed specification sheets and drawings are considered as the documents produced by the heat exchanger group, and valve, pipeline calculations, connection information sheets, etc. as the information provided by the systems group. The characterization of information was provided by the Pullman-Kellogg Company in the form of document blanks and through discussions with one of the authors.

As a first step in using database management technology, modelling the existing communication system

*Department of Chemical Engineering, University of Texas, Austin, Texas, USA
†Department of Computer Science, University of Texas, Austin, Texas, USA

 0010 4485/79/0121 -06 $02.00

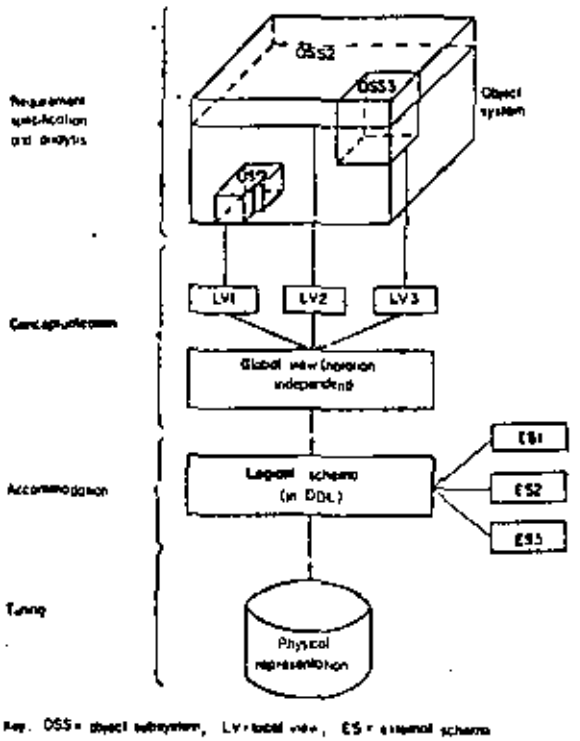


Figure 1. Database design process

which has proven to be adequate for engineering purposes is proposed. The following examination of the problems of database design methodologies is made in the context of this application.

DATABASE DESIGN PROCESS

The term 'database' is an elusive one. In this context a database is considered to be a model of a portion of reality represented by an evolving collection of interrelated data accessible to several applications. These data have a controlled redundancy and the user has the capability of inserting, deleting, updating and retrieving data in a restrained form through the actions of a database management system (DBMS).

The extraction of all the characteristics of the object system which is to be represented in the database is the first step in any database design, and initially involves user requirements specification and analysis. The representation of the object system's properties and their integration into one conceptual model is the next portion of the logical database design. The model obtained by this process is an abstract model which represents the inherent properties of the object system and is independent of any model or notation supported by the DBMS. In order to be processable the conceptual model has to be expressed in terms compatible with a particular DBMS. This is the database accommodation step. Since local views were integrated at a high level of abstraction to yield a binding model common to all applications, it is necessary to represent now the user's views in terms of the data description language (DDL) supported by the particular DBMS. The whole process, from determination of user requirements through conceptual model and final accommodation in terms of the selected DBMS, is the logical database design process.

The internal organization of the data representing the object system and its models on physical devices constitutes the physical design of the database. Figure 1 shows schematically a view of the database design process.

The strong effect that any misconception of the object system or misrepresentation of its characteristics in the conceptual model has on the ability of the database to satisfy the users' information needs has centred attention on the logical database design stage. The systematic process by which one traverses the different stages of the logical database design and performs the mappings from one level of abstraction to the next is commonly called a logical database design methodology.

Two parameters have a strong influence on the complexity of database design:

- the number of objects to be represented,
- the number of interrelationships amongst objects.

The area above the curve in Figure 2 represents the area in which intuitive design becomes most difficult. To model an object system in which many objects and activities are interrelated a systematic and well-structured approach is necessary. It is therefore instructive to observe how proposed methodologies perform in a large-scale application, what criteria should be applied for evaluation of existing data modelling techniques, and what consequences can be drawn for the development of future ones.

Since database design techniques have evolved largely for the purpose of supporting business applications it is useful to analyse the capabilities of various methodologies in the technical design environment described in the previous section. The remainder of this paper will therefore discuss observed strengths and weaknesses of the examined methodologies as a basis for defining a set of practical criteria for evaluating a methodology. Since no examined methodology fulfills all the criteria established, the final section states the need for the synthesis of a new methodology responsive to the postulated criteria. As the term 'synthesis' indicates, this should not be a complete redevelopment, but an assembly of the best parts of already proposed

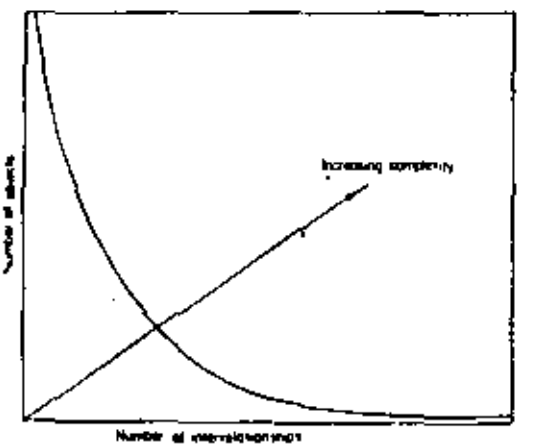


Figure 2. Boundary of feasibility for intuitive logical database design

design techniques integrated into a framework capable of bridging existing methodological shortcomings.

METHODOLOGIES ANALYSED

Database design methodologies with markedly different characteristics and origin are examined in order to cover a wide spectrum of features. The selection under consideration is limited but is a fair representation of the tendencies noticeable in research in this field. Two basically different trends can be observed. One approach first generates a global schema and then derives local views from it; this approach is characteristic of much current work in Europe. More popular in the US is an approach which first models local views of different users and then integrates them to form a global view. Modelling approaches that assume different data models, and which have the potential for future extension and refinement were also selected.
The result was the selection of 3 methodologies, namely: Bubenko[2,4], Kahn[5,6] and Smith and Smith[7–9].

Space limitations do not allow an extensive review of these methodologies, and it is suggested that the original work be consulted for further details. Only the salient characteristics and specifically those which will influence the evaluation criteria will be cited.

Bubenko's methodology

Bubenko's work has been evolving and the comments made here apply basically to the version presented in the Inferential Abstract Modelling (IAM) approach[1,4]. The process can be divided into seven iteratively executable groups of activities:

- collection and specification of information requirements
- entity classification
- specification of functional dependencies
- abstract object specification, integration and analysis
- implied information analysis
- derivability (precedence) analysis
- transformation to an external name-based model

The first step consists mainly in a narrative gathering of information requirements and their formulation in terms of queries and transactions. The second step, the classification of entities into concept classes is not further formalized and appears to be an extraction of all the subjects and objects (grammatically speaking) of the query and transaction descriptions. The results of this step are highly dependent on the care and degree of standardization with which the requirement formulation was carried out. It is conceivable that information is lost or strange entities are added because of poor requirements description by the users. An example occurs in the engineering design application context where document names are well-understood concepts for a designer. In this case, it can happen that query or transaction requirement statements will be formulated in these terms, thereby introducing a report format into the conceptual model as an object. Any change in reporting procedure would imply a change in the conceptual model.

The grouping into concept classes proposed by Bubenko is highly intuitive and not likely to be formalized because of the application dependence of this step. However, guidelines would be helpful.

The third step, the determination of functional dependencies, is a monumental task in a semantically rich application. The IAM methodology recognizes various kinds of functional dependencies. In the analysed context some examples could be:

F(1): tubesheet — component of → heat exchanger
F (2): equipment function ← - id - → equipment identifier
F(3): flowrate, ΔP_{max} — d → pipe diameter
F(4): composition, temperature, pressure ← — mat → valve material

Several problems are encountered in an engineering environment, the most difficult being who should define the dependencies. A database designer cannot be expected to be familiar with every aspect of an application and hence cannot discern all possible dependencies. The user, on the other hand, has a limited perception of the whole problem space. In addition, some functional dependencies [e.g. F(3) and F(4) above] represent design decisions and some dependencies can be expected to change from project to project.

The abstract objects are determined as a result of the previous steps and it may be necessary to iterate. It should be noted that this approach uses two levels of abstraction. Only later are the abstract objects mapped into a name-based model. The implied information analysis comprises the identification of implied and derived associations and the corresponding rules. An association is implied if it is functionally dependent on a proper subset of the initial or identifying associations related to a particular abstract object. Identifying associations are those that cannot be removed without destroying the 1:1 correspondence property. Associations are considered derivable if they are fully dependent on the set of identifying associations. The precedence or derivability analysis serves as a test to ensure that the initial abstract objects and the defined rules are sufficient to satisfy all the anticipated information requests. This abstract model is mapped in the last step into a name-based model through the introduction of name sets.

One of the interesting features of Bubenko's approach is the inclusion of a time dimension, but this characteristic appears not to be operational. A strong point of the methodology is the separation of an abstract and a name-based level. The use of inferred and deduced information provides a possibility for checking for consistency at the logical level, since various paths are available to obtain certain information, but problems are the implementation level could rise.

Kahn's methodology

Kahn's methodology appears to be a good candidate to serve as a framework. A characteristic of this approach is the early separation of the problem into two parallel perspectives, the information structure perspective, which describes the interconnections within an organization, and the usage perspective, which is concerned with satisfying the processing requirements within an organization, and amalgamated these two perspectives should provide the best overall logical design. The separation into two perspectives raises one important issue: should two different extreme schemata be designed, one processing, the other information oriented, which are merged at the end of the logical design to yield the conceptual schema of the enter-

prise or should the design of one, very flexible, information structure first be undertaken, introducing through a gradual refinement those modifications that allow the processing requirements to be fulfilled? An analysis of the evolution of Bubenko's and Kahn's work in the last two years shows a tendency towards the second approach, but neither author makes a strong argument for either alternative. Kahn's methodology basically consists of 6 levels and 5 steps to go from one level to the next:

Levels	Steps
Real world requirements	
	Requirements step
Local information structures	
	Entity step
	Relationship step
Global information structure	
	Entity structure step
Entity structure	
	Refinement step
Revised entity structure	
	DBMS accommodation step
Logical database structure	

One of the strengths of this methodology is the detailed specification of the input/output for each step, the major shortcoming is the lack of detail in the available papers as to how one should proceed to obtain part of this output.

The requirement step is subdivided into 5 activities: a gross system analysis; a selection of design path between information structure perspective and processing requirements perspective with the former being tacitly preferred; a requirement technique selection; requirement collection and specification; a requirement analysis to produce individual information requirements documents. However, little information is given as to how these documents are produced.

Kahn's methodology aggregates the local views into a community view and insists that each user be responsible for his requirements, thereby avoiding users anonymously stating unreasonable requirements. The entity step serves the purpose of designing global entities which are representative of the organization's aggregated needs. An analysis of the activities performed indicates that this methodology arrives directly at a name-based model. The aggregation to form a nonredundant collection of entities is one of the more difficult tasks and is left largely to the designers' intuition. An equivalent step is carried out for relationships creating global relationships, identifying conditional and existent relationships and eliminating all redundant relationships. It is important to realize that a strictly nonredundant global information structure is obtained here which might be considered as the initial conceptual schema.

In the entity structure step, the global information structure is subsequently mapped into functional dependencies and from there into normalized relations. The published papers on this methodology do not describe what specific algorithm is used for the synthesis of normalized relations. However, according to Bernstein[16] all known algorithms depend on two assumptions: the availability of a set of all functional dependencies, and their uniqueness, i.e. that, at most, one functional relationship connects one set of attributes to another. The treatment of functional dependencies in the published algorithms for construction of normalized relations is strictly syntactic and, therefore, the algorithms not only depend on a complete set of functional dependencies, but on a complete set of functional dependencies that cannot lead to invalid syntactic inferences. Kahn's methodology tries to circumvent this problem by resolving the semantic ambiguities in previous steps. By obtaining the functional dependencies from the global information structure rather than directly from the user requirements it is more likely that the strictly syntactic approach will be successful. However, the FDs still have to be checked for compliance with the uniqueness requirements and the necessary axioms for construction of relations. If Bernstein's method is used, these axioms would be reflexivity, augmentation, and pseudotransitivity.

Once in the form of normalized relations, the refinement introduces controlled redundancy, ensures adequate processing and accommodates security and volatility of the data. However, it is possible that some of the actions described in this step will be used for garbage collection, grouping unresolved issues and putting them out of sight.

The DBMS accommodation step comprises all those actions necessary for expressing the conceptual schema in the DDL of a chosen DBMS and requires that a mapping be possible from normalized relations to the model supported by the selected DBMS.

Smith and Smith methodology

The methodology introduced by Smith and Smith ignores the requirement gathering and analysis stage and assumes that the concepts to be integrated are available and are fully understood by the database designer. The basic premise of this work is that abstractions are the means by which humans understand and manipulate complex systems. For database design only abstract objects are considered of interest and they can be formed through generalization and aggregation. An object is formed through generalization from a class of other objects ignoring the differences in favour of the common properties. An abstract object is formed through aggregation by naming a relationship among other objects. Repeated application of generalization and aggregation to the set of names provided by the requirement step results in abstraction hierarchies, which can be visualized as lying in perpendicular intersecting planes. In this methodology the names carry a large portion of the semantics. The authors contend that it is inappropriate to assign a fixed interpretation to an object at the conceptual level. An object can be viewed as entity, component, relationship, category, attribute, or instance, depending solely on the viewpoint of the user. It is claimed that set theoretical models are less restrictive in this sense than graph theoretical models which tend to fix the interpretation. If this is the case, it will be necessary to analyse what restrictions have to be imposed in order to map the conceptual schema obtained through abstractions into a graph oriented DBMS – an important step since graph (network) oriented DBMSs are commercially available.

The methodology consists of few basic steps: generalization, aggregation, instance identification and expression in terms of an abstract syntax. The activities of the abstraction steps are highly intuitive and depend on the insight and abstraction capacity of the database designer.

IDENTIFICATION OF CRITERIA

In the preceding discussion, the salient characteristics of some existing methodologies were identified. In this section a framework for evaluating database design methodologies is outlined and the desirable features of a specific methodology proposed for the area of interest are identified.

Database design methodologies may be characterized by the following tentative list of properties:

- completeness
- design strategy
- requirement specification procedure
- number of abstraction levels
- aggregation process
- separation of information and processing requirements
- scope of the model
- use of functional dependencies
- use of roles
- treatment of security, update propagation, etc.
- modularity
- adaptability to automated design tools

The completeness of a methodology, incorporating smooth transitions from one step to the next and adequate mapping facilities to go from one level of abstraction to another is a necessary condition for acceptability. Gaps in proposed methodologies frustrate practising database designers and lead to *ad hoc* extensions.

Design strategy for software is often discussed in the form of the debate 'top down vs. bottom up'[12]. The labelling of a methodology as top down or requirements first is often confusing. For example, an early version of Bubenko's methodology is referred to once as top down[11] and elsewhere as requirements first[13]. In the case of database design methodologies the debate can be reduced to the question 'what is the initial image of the object system?'. While requirements first methodologies relate directly to the physical object system, top down methodologies use a preconceived mental model of reality which will, most likely, be incomplete and biased. For engineering purposes a requirements first approach is more adequate.

If a requirements first logical database design methodology is advocated, it is necessary to ensure that a good requirement gathering - analysing procedure is part of the selected methodology. The need for this is recognized by most authors but is carried through only by a few. Frequently emphasis is put into the development of requirement specification languages and automated analysis procedures, which are helpful only when the physical reality has been correctly perceived. The problems which are most difficult to eliminate arise from false perceptions of reality, resulting from a natural language barrier between the 'database naive' user and the 'application naive' database designer and the lack of appropriate tools for communication between them.

Most methodologies, having their origin in a business environment, do not consider the major source of information available in engineering environments, the standardized design documents. Each document represents the local view of a small portion of reality and is therefore an excellent starting point in gathering information about that portion of the object system. This approach has proven useful in the application considered here and has the added advantage of being very economic as far as users' time is concerned.

Some of the analysed methodologies include two abstraction levels, while other include one. When two levels are used, the naming procedure is usually delayed and first the physical reality is modelled in terms of abstract objects. Naming occurs at a lower level of abstraction, i.e. a later stage in the design methodology. If only one level of abstraction is provided, the database designer arrives immediately at a name-based model which is more restrictive and may be more difficult to manipulate because of the previously mentioned language barrier and the semantics conveyed by the names. The high semantic content of names makes the assumption that the database designer understands the unique meaning of each name and can resolve any naming conflict questionable. This suggests the use of two levels of abstraction as the preferred approach.

A major difference among existing methodologies is their approach to the aggregation process. Such methodologies aim at a single conceptual model, spanning all the applications in one step. Other methodologies model each application separately in the form of local information structures and later integrate them into a global view. For large engineering applications the second approach appears to be better, but whatever the number of aggregation steps may be, except for the most trivial applications, some support is necessary whenever overlapping of entities and attributes exist. It is this point where most methodologies fall short and leave the problem to the designer's intuition.

Methodologies vary in their consideration of processing requirements. There appears to be a consensus that the information perspective should be given priority. Once this is obtained processing requirements can be introduced either by defining an extreme, processing oriented schema and amalgamating it with the information oriented schema, or a gradual refinement of the information requirements-based schema can be attempted. The latter offers several advantages. Changes are gradual and need only be carried out until the required performance is met, thereby avoiding an overshooting of the performance requirements at the expense of flexibility. However, it is necessary to have a good monitoring mechanism to measure the effects of a given modification.

The scope of the model determines the degree of redundancy at the conceptual level, i.e. should information be stored explicitly or should as much as possible be derivable. If little information is stored explicitly and many inference rules are established there may be alternate ways of deriving desired information, thereby allowing the designer to verify correctness at the conceptual level. There is, however, a trade-off which has to be considered, especially in complex environments. In order to obtain correct inference rules the functional dependencies have to be known and this may be difficult. It can be anticipated than any methodology depending on specifying a complete set of functional dependencies will not be suitable for the design of a complex engineering application database. Emphasis on inference of information may also result in conflicts with the users in our area of interest since many engineering design decisions

are inferences based on the data stored in the database. Therefore extensive use of inference can be part of a database system supporting local workspaces, but not in a system used as the official record of a project.

The varying flexibility with which methodologies treat individual data items, either assigning a fixed role or allowing free interpretation depending on the particular view has to be considered when synthesizing a methodology. It is not feasible to integrate portions of methodologies which differ in their perception of the role of an item.

Before selecting a methodology it is necessary to ascertain if there are any unresolved steps that may play a crucial part. Issues like security, and update propagation are often found in 'garbage collection steps' towards the end of the methodology, but they may influence early stages in the logical database design and should be identified at the policy definition step to be resolved, if necessary, as the design progresses and avoid costly iterations and redesigns.

Modularity is a special concern in relation to the problem of synthesizing a methodology, but it is also necessary when trying to carry out some tasks in an automated or computer-aided form. Design tools are seldom comprehensive and clearly defined stages with well-specified input and output are necessary for a design aid to be useful.

CONCLUSIONS

Some existing methodologies have been outlined and some of the characteristics discussed. This resulted in a series of practical issues to consider in constructing a logical database design methodology for application in an engineering design environment. For the environment under consideration all the screened methodologies presented shortcomings. Consequently we are currently engaged in synthesizing a methodology with the following characteristics:

- a requirements first approach is utilized, even though this carries some risk of bias in favour of exising applications and problems
- the methodology will be comprehensive, i.e. it will include as integral parts (a) the mapping from reality in the form of defined requirement gathering procedures and (b) the mapping to a chosen DBMS in the form of a database accommodation step
- the requirement gathering procedure will make extensive use of engineering documents
- issues like security and propagation of updates will be defined at early stages and included in the requirement gathering/analysis procedure
- guidelines for identification of entities and attributes will be generated
- two levels of abstraction will be incorporated
- algorithms or heuristics for support of the aggregation process will be part of the methodology
- a conceptual schema that stresses the information requirements is favoured, with introduction of processing requirements through gradual refinement. Refinement will occur upon mapping into a DBMS dependent representation
- specification of obvious functional dependencies will be made, but it is recognized that a complete set is very unlikely to be obtained. Inference rules for deriving information will be avoided in favour of explicit storage whenever inference would interfere with the users' decision process
- the methodology will be modular in order to be able gradually to adapt automated design tools

ACKNOWLEDGEMENTS

The authors thank the Pullman-Kellogg Company for providing data used in the present research, and particularly Mr Alex Myles, Mr Carl Albers and Dr Burnelle Landry for their active cooperation. Thanks are also expressed to Professor Janis A Bubenko Jr for his interest and useful comments.

The present research was partly sponsored by Consejo Nacional de Ciencia y Tecnologia, Mexico.

REFERENCES

1 Novak, D O and Fry, J P 'The state of the art of logical database design', *University of Texas Conf. on Computing Systems*, Austin, Texas (Oct. 1976)

2 Yao, S B, Navathe, S B and Weldon, J L 'An integrated approach to logical database design', *NYU Symp. on Database Design*, New York City (May 1978)

3 Bubenko Jr, J A 'IAM: Inferential Abstract Modeling — an approach to design of large shared databases', *IBM Res. Rep. RC6 343*, Yorktown Heights, NY (June 1977)

4 Bubenko Jr, J A 'IAM: An Inferential Abstract Modeling Approach to Design of Conceptual Schema', *ACM SIGMOD Int. Conf. on Management of Data*, Toronto (August 1977)

5 Kahn, B K 'A method for describing the information required by the database design process', *Proc. SIGMOD Conf.*, Washington DC (June 1976)

6 Kahn, B K 'A structured logical database design methodology', *NYU Symp. on Database Design*, New York City (May 1978)

7 Smith, J M and Smith, D C P 'Database abstractions: aggregation', *Commun. ACM* (June 1977)

8 Smith, J M and Smith, D C P 'Database abstractions: aggregation and generalization', *TODS* (June 1977)

9 Smith, J M and Smith, D C P 'Principles of database conceptual design', *NYU Symp. Database Design*, New York City (May 1978)

10 Bernstein, P 'Synthesizing third normal form relations from functional dependencies', *TODS* (December 1976)

11 Bubenko Jr, J A, Berild, S, Lindencrona-Ohlin, E and Nachmens, S 'From Information Requirements to DBTG — Data Structures', *Proc. ACM-SIGMOD-SIGPLAN Conf. on Data: Abstraction, Definition and Structure*, Salt Lake City (March 1976)

12 Bubenko Jr, J A 'Validity and verification of information modeling', *Conf. on Very Large Databases*, Tokyo (October 1977)

CHAPTER 3

DATABASE ARCHITECTURE, CONCEPTUAL MODEL, AND DATA DICTIONARY FUNCTIONS

3.1 Database architecture

Databases evolved from simple file handling systems as the need arose to cross-link those files in order to access data for other applications. At that stage database design was considered solely the specification of records and their organization on secondary storage devices. As a result the user had to be aware of many storage-related aspects. In an attempt to improve data independence and to structure the database design process, in 1971 the CODASYL database task group (DBTG) proposed a two-schema database architecture (DBTG71), which was followed in 1975 by the three-schema approach presented by the ANSI/X3/SPARC study-group (ANSI75,JARD77). In order to achieve a well structured database it is necessary to decide first on an appropriate architecture. Therefore, this chapter is devoted to a brief review of the DBTG and the

ANSI/X3/SPARC proposals, a description of the topics relevant to this research which need further specification, followed by a discussion of the restrictions to be imposed on the ANSI/X3/SPARC architecture for practical reasons. Three topics of interest are intimately related and each is discussed as a function of the other two: the conceptual schema design process, the model to be used in the formulation of the conceptual schema, and the function of the data dictionary during conceptual schema design.

3.1.1 The CODASYL/DBTG architecture

The proposal of the CODASYL DBTG was a first attempt to standardize database design and group the user-related aspects in the various subschemata, while the common view and all the storage related aspects were specified in the schema. The separation, however, was by no means complete, as can be evidenced by many constructs in the CODASYL data description language (DDL) and data manipulation language (DML) (DBTG71). The two-schema approach, although it is the basis for several commercially available sytems has the following drawbacks:

1) the database design is not transparent to the user, i.e. the user has to be aware of accessing mechanisms which are specified at the external schema level;

2) changes in the storage strategy affect the users, even in those cases where the logical content of the database is unaltered;

3) systems designed according to the CODASYL proposal are locked into the DBTG data model, a restricted network model;

4) the main aim is efficient operation at the expense of flexibility in the logical design.

3.1.2 The ANSI/X3/SPARC database architecture

In order to overcome the shortcomings of the CODASYL architecture the ANSI/X3/SPARC study-group set out to formulate a database architecture which would allow complete data independance. This would be achieved using three schemata: a set of external schemata which represent the users' view of the data, very much like the CODASYL/DBTG subschema; a conceptual schema, which is a global, unbiased view of the whole enterprise and which is independent of the various user views as well as the

physical storage structure; and an internal schema, which is a blueprint of the physical storage structure.

The ANSI/X3/SPARC study-group did not attempt to present a database system which was ready for implementation and would henceforth serve as the only standard, but rather tried to identify the necessary components of a database architecture which would guarantee a maximum of flexibility and data independence and would represent a goal to be achieved through further refinement. In this spirit the database system was broken down into logical components and interfaces or mappings needed for communication between connected modules. Only the section of interest to the present research is shown in figure 3 (YORB77). It represents the interplay of the three schemata, conceptual, external, and internal, the data dictionary/directory (DD/D) and the necessary interfaces.

The truly innovative part is the conceptual schema, which isolates the physical storage and access mechanisms from the user views, thereby providing the data independence sought. In the most general case different data models could be supported at the external level which would be mapped back and forth to and from the conceptual schema. This architecture received the

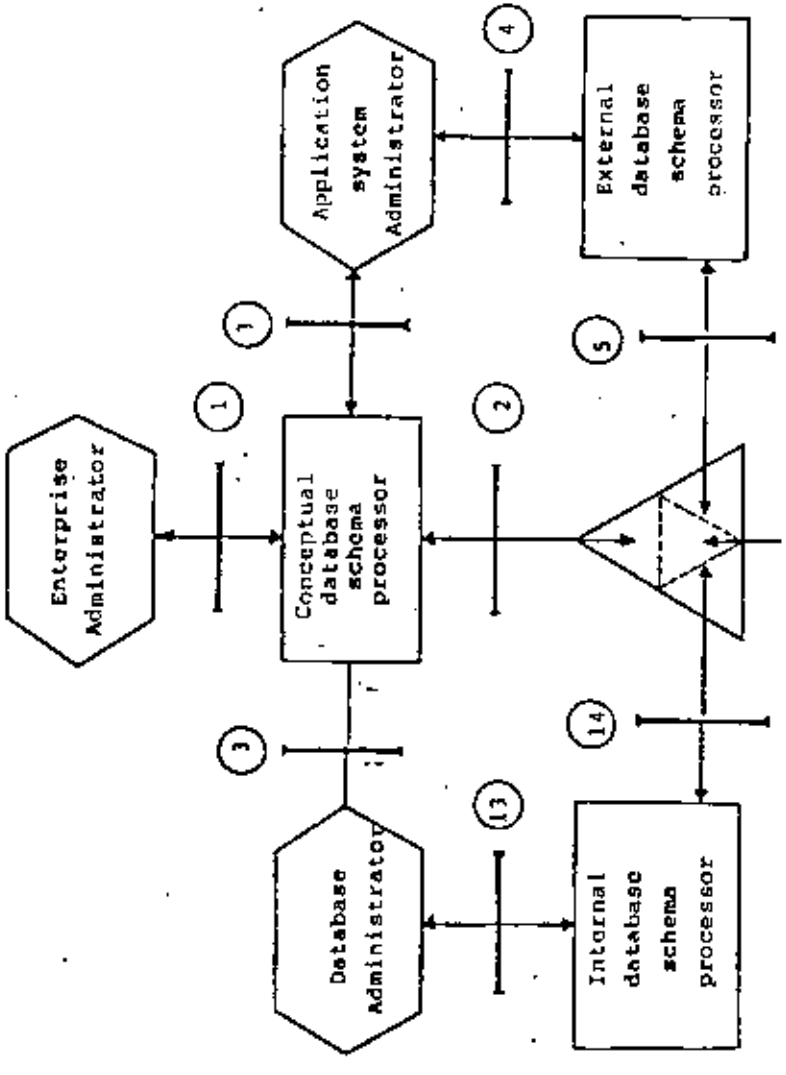


Figure 3. The relevant portion of the ANSI/X3/SPARC architecture (YORB77)

name of coexistence architecture (NIJG76).

The ANSI/X3/SPARC report also specifies the roles of the enterprise administrator as the person ultimately responsible for the conceptual schema, of the database administrator as the individual(s) responsible for the internal schema and the application administrator(s) in charge of the formulation of the external schemata, thereby setting the necessary organisatorial framework.

The ANSI/X3/SPARC architecture represents progress in the right direction, but the study-group was not chartered to resolve all the existing problems. It appears at the present time that the ANSI/X3/SPARC architecture will prevail and that any system with a long-term scope will have to adapt sooner or later to the ANSI/X3/SPARC proposal. However, many questions are still unanswered and must be solved before the ANSI/X3/SPARC architecture is operable in its most general formulation. In the following sections the necessary restrictions will be specified in order to achieve the most flexible architecture possible under present constraints. The architecture presented here is based on the ANSI/X3/SPARC proposal with restrictions that can be gradually eliminated as ongoing research shows how to handle the necessary mappings. Once the

restrictions are eliminated, the present architecture will have evolved into the one proposed by ANSI/X3/SPARC.

3.2. Areas in the ANSI/X3/SPARC architecture requiring further specification

Three areas of concern will be discussed here: the interface between the enterprise administrator and the conceptual schema, the mappings between the conceptual schema and the various external schemata, and the role of the data dictionary/directory.

Interface 1 in figure 3 is the interface through which the enterprise administrator defines the entities of the real world (or object system) and their relationships in terms of the conceptual schema. It is, in fact, the conceptual schema design process. No attempts were made by the ANSI/X3/SPARC committee to formalize this procedure. With the conceptual schema being the core of this architecture, the lack of a formal definition of the database design process is perhaps the most serious shortcoming of the proposal. A review of some attempts to fill this gap is given in the next chapter, and a proposal to define this logical design process for engineering systems is the theme of this research.

The DD/D is defined in the ANSI/X3/SPARC report as the repository for the machine readable form of the conceptual schema upon which the external and the internal schema processors operate. However, the lack of a formalization of the design process makes it impossible to specify the functions of the DD/D during the design of the conceptual schema, one of the functions of the DD/D which could prove most productive.

Finally, no consensus has been reached among researchers about the model to be used for the conceptual schema. This is due, in part, to the dual role of the conceptual schema in design and operation of the database. The selection of an adequate model deserves, therefore, special attention, since many of the features of the database design process and the DD/D will depend on the model used.

3.2.1 Necessary restrictions to the ANSI/X3/SPARC architecture

The conceptual schema in the ANSI/X3/SPARC proposal serves a dual purpose:

1) it is a central, unbiased description of the enterprise common to all users and during the database

design it is the global description from which the external and internal schemata are derived;

2) it is also an intermediate representation in the run-time mapping from the different models supported in the external schemata to the internal representation of the data.

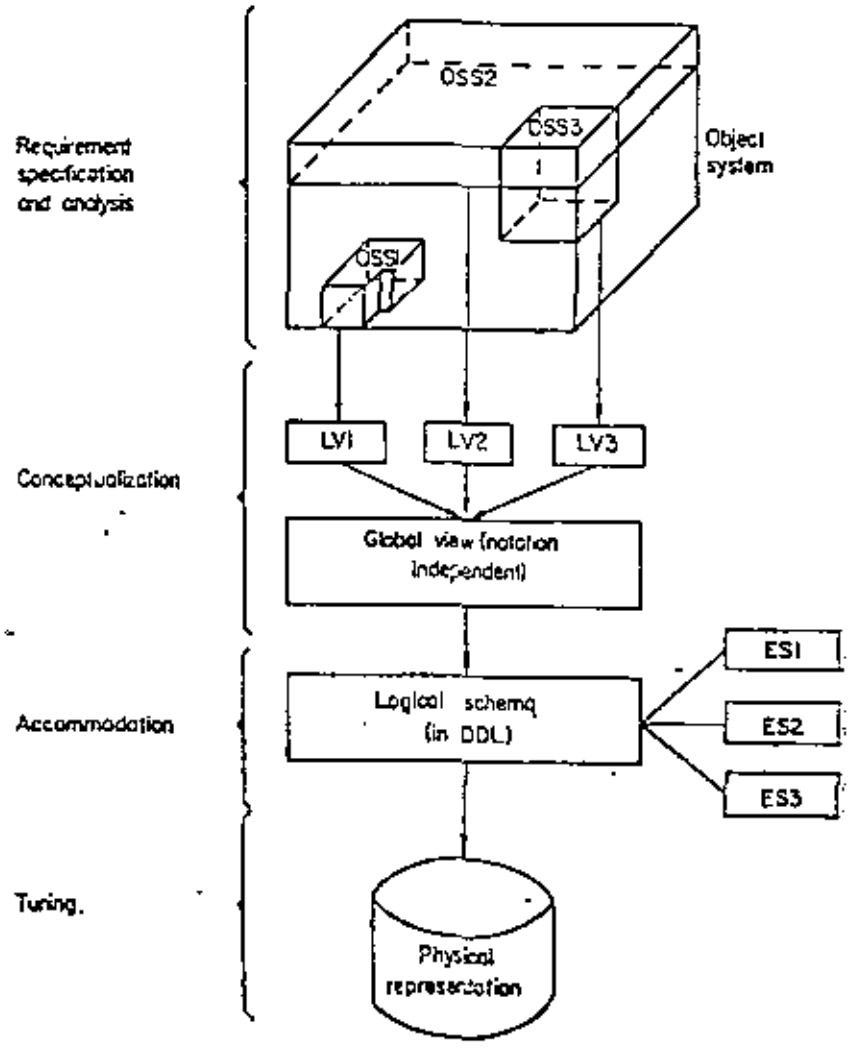
The conceptual schema in its first role is currently attainable as the result of the logical database design process. This is in fact the static portion of the conceptual schema concerned with the structure of the database without consideration of the values. The dynamic role of the conceptual schema is not attainable at present since it has not been demonstrated that the two-way mapping between the conceptual model and the models supported by the external schemata is possible without the loss of semantic content and/or introduction of ambiguities in the execution of the update and insertion operations.

Since models which are best suited for conceptual modeling are not those supported by the majority of available DBMSs the logical conclusion is to divide the conceptual schema into two parts: a global view of the enterprise in the semantically most explicit model, and a

logical schema in the model and the DDL of the DBMS which is to operate the database. The global view can eventually perform all the functions of the conceptual schema once the necessary mapping processors have been implemented, provided it has been established under which conditions the necessary mappings can be carried out unambiguously, and that an adequate model has been used for the definition of the global view. Henceforth, the terms 'global view' and 'logical schema' will be preferred over the more general 'conceptual schema'.

The second major restriction, which is a consequence of the first, is that only subsetting is allowed as the mechanism to define the external schemata.

The logical database design can then be introduced into the architecture of figure 3 as depicted in figure 4. Here interface 1 consists of the modeling of portions of the object system or enterprise to obtain the corresponding local views. A second mapping, actually a merge of all the partial views, results in a global view of the enterprise in terms of the semantically most explicit model, from where a third mapping produces the logical schema, which is the global view expressed in the DDL of a particular DBMS. From the logical schema the external schemata can be defined by



Key: OSS = object subsystem, LV = local view, ES = external schema

Figure 4. Schematic representation of the database design process

subsetting and the necessary mappings to the internal schema will include definition of the storage and access mechanisms.

3.3 Selection of a model for the global view

For the representation of local and global views a model is sought that fulfills the following conditions:

1) have the capability for describing the reality or object system with all its semantic characteristics;

2) be stable to changes in the object system;

3) be flexible by minimizing within the model the number of categories , such as entities, attributes, relationships beteeen entities, and relationships between attributes;

4) allow mapping to models used in commercial DBMSs;

5) allow modeling of the object space subject only to information requirements without inclusion of access conditions to separate clearly the logical from the physical design.

The candidates to be considered (grouped into major categories) are the hierarchical, network, unrestricted n-ary relational, irreducible n-ary relational, and binary associative models. For an extensive taxonomy the papers by Kershberg, et.al. (KERL77), Biller and Neuhold (BILH77) and the proceedings of the IFIP 2.6 meeting in Freudenstadt (NIJG76a) should be consulted.

3.3.1 The network model

The network model distinguishes between entities and their attributes. Relationships among the elements of the object space can either be associations among entities, or record types which define the relationship among attributes of an entity set. The network model allows for two types of links between entities, those which carry information and are therefore essential to preserve the semantic content and those which do not. A link is information carrying if it is the only means by which the relationship between two entities can be assessed. On the other hand if an attribute of one of the entities involved in the association carries the same information as the link, this link is not essential and is not considered an information carrying link.

The major shortcomings of the network model are the classification of data into entities and attributes, which is often conflicting when modeling different applications that will be integrated. As Tsichritzis points out (TSID77) associations and attribute relationships are syntactically not very different, and they can be difficult to distinguish. The difference between information carrying and non-carrying links illustrates best how the semantics are not always stated.

Because of these characteristics the network model is not readily adaptable to changes in the object system and semantic information may be lost or misinterpreted when modifying the global view in response to changes in the object system.

3.3.2 The hierarchical model

The hierarchical model (TSID76) may be viewed as a restricted version of the network model where each descendant can only be connected to one parent and where crosslinks between descendants are not allowed. This restriction, although efficient for processing, is too rigid for logical modeling. Since only one link is allowed between two objects it is not necessary to name the links, thereby embedding the semantics heavily into

the access structure. This in turn makes any schema defined in terms of a hierarchical model unstable and unresponsive to changes in the object system.

3.3.3 The unrestricted n-ary relational model

In the unrestricted n-ary relational model as proposed by Codd (CODE70) data are organized into relations. A relation R is defined as a subset of the Cartesian product of its domains:

$$R \subseteq D1 \times D2 \times \ldots \times Dn$$

This results in a table in which rows represent the tuples of the relation and the columns the occurrences within each domain.

In the relational model the representation of information is unified, i.e. distinctions of information carried in an information carrying link or as an attribute, as in the case of the network model, are eliminated. However, by allowing a relation to be built over an arbitrary number of domains some of the semantic content is lost, since semantics are expressed basically in the relation's name and through association (used in colloquial form) of the domain roles. The problem can be alleviated by normalization [1]. However, some problems

arise in this case. Carrying out the normalization to have relations in 2NF can result in relations having domains which are transitively dependent on each other. To avoid this, further normalization to 3NF is required. It is then necessary to know and follow all the functional dependencies underlying that relation. A problem of 3NF relations based on functional dependencies is that some of the resulting relations may not represent meaningful facts from the object system. This requires the association of domains over different relations by matching of values in the common domains.

3.3.4 The irreducible n-ary relational model

The use of irreducible n-ary relations is an attempt to overcome the loss of semantics associated with a grouping of too many domains in one relation and, at the same time, to combine all the domains which are essential to preserve the meaning of a fact in the real world in one construct. Therefore, an irreducible n-ary relation can be understood as the subset of the Cartesian product of exactly as many domains as are essential for representation of one fact as one relation. For example,

[1] In order to clarify the functional dependency and normalization concepts, definitions and an example are given in Appendix A. Further reading on normalization can be found in DATC77, CODE70, CODE74.

date requires domains day, month, and year. A more formal definition of irreducibility is given by Hall, Owlett and Todd (HALP76). According to them, a relation is irreducible if it cannot be broken down by projections into several relations of smaller degree in such a way that they can be joined to reconstitute the original relation [2]. The problem with this definition is the dependence on instances, i.e. values, in order to establish what relation is or is not irreducible. At the database design stage all possible instances cannot be foreseen and, therefore, this criterion is not very helpful. At the same time it illustrates the difficulty involved in determining what an irreducible n-ary is.

Irreducible n-ary relations have been recommended by Falkenberg (FALE75,FALE76a,FALE76b,FALE77) and Hall et. al. (HALP76) as the basic constructs for logical modeling and are introduced into Falkenberg's object-role model (FALE75,FALE76a). In this model each object has a role associated and two object-role pairs are considered a 'binary association'. This construct is, strictly speaking, a binary relation if one considers an

[2] Incidentally, this definition is Date's description of normalization from 1NF to 2NF (Date, op. cit. page 160). Similar observations regarding irreducible n-ary relations in their present definition were made independently by Biller (BILH79).

association to be unidirectional and a binary relation to consist of two binary associations (RAVN77). Similarly n-ary 'associations' can be built by grouping n object-role pairs. The object-role model allows an association to act as an object thereby permitting the nesting of binaries, but introducing a dual character for associations.

Proponents of the irreducible n-ary models argue that irreducible n-aries can be obtained through an algorithm from unrestricted n-aries. However, Delobel's comments at the Freudenstadt meeting (DELC76) suggest that such an algorithm can only function if additional information is provided in the form of binary associations. This is particularly so when irreducible n-aries are built by nesting binaries. Furthermore, it has been shown by Senko (SENM76a) that irreducible n-aries are not necessarily any easier to manipulate and understand than binaries, especially in the case of large interactions. For example, two ternaries with two common domains require six identifiers, while overlapping binaries only require five.

3.3.5 The binary associative model

Binary associative models for the conceptual schema have been proposed by Senko (SENM75,SENM76a,SENM76b) and Bracchi et.al.(BRAG76) among others. They do not distinguish between entities and attributes and different kinds of associations. The only elements of this model are the objects and associations between pairs of objects. Since any categorization into entities and attributes is avoided, no bias derived from a particular user-view is introduced. Each association has to be named, and the association name can carry the semantics of each association. Since these binary associations are the smallest possible constructs, they can be considered as the semantically most explicit model.

Local and global views are, when expressed in terms of this model, basically collections of triplets formed by an association name, a subject (the object the association originates from) and a predicate (the object the association points to). Addition of associations, which accounts for the major portion of changes in the conceptual schema, can be accomplished by insertion of the new triplets. Simplicity and ease to evolve are two of the main advantages of the binary models.

While binary associative models impose restrictions on the modeling process, these restrictions, as pointed out by Bracchi et.al. (BRAG76), are specified by the enterprise administrator as a function of the interacting applications and not, as in the case of functional dependencies, by the model itself.

Because of their simplicity the binary models have been attacked as 'primitive'. The main argument is the need for 'artificial objects' in order to model more complex facts. This is only a disadvantage in extremely simple cases where only few n-aries can span the whole universe of discourse. In the case of a conceptual schema where the binaries can be expected to be the building blocks of many n-aries the internal objects actually ease the handling of complex schemata (SENM76a). Especially in the integration process of several partial views it is much easier to handle the minimal units rather than large and rigid n-aries. Bracchi also justifies the creation of internal objects (internal sets of concepts in their nomenclature) over the nesting of binary associations. Expressing a ternary or n-ary concept as nested binaries is an arbitrary process, since any initial pairing could be acceptable, for example, (day-month) - year, day - (month-year), or month -

(day-year).

The other issue against binary models, stating that binary models are less 'natural' than n-ary models, can be reduced to a matter of preference based on experience in handling one or the other type of construct.

Finally there are some considerations of a practical nature. Existing computer aided database accommodation packages, such as IBM's Data Base Design Aid (DBDA) (IBMC,RAVN77), start from binary associations in the creation of schemata in the DDL of a particular DBMS. This is a substantial advantage given the present architectural constraint which forces the separation into a global view in the conceptual model and a logical schema in the model and DDL of a commercial DBMS.

Last but not least there is the availability of a working binary associative DBMS, CS4 developed by the CADIS group at the University of Stockholm (JANM79,BERS77a,BERS77b). This system was provided by Prof. J. A. Bubenko Jr. and is now operable on the DEC-10 at The University of Texas at Austin and is used for testing of partial and integrated views, as well as the implementation of a binary associative data

dictionary.

3.3.6 Conclusion on the model selection

While the selection of a relational model over hierarchical or network models is a clear choice, the different relational or associative models all have proponents. It is considered, however, that the balance is favorable to binary models on the grounds of:

1) simplicity in the formulation of associations;

2) semantic clarity which allows only explicit associations between objects;

3) flexibility and absence of user bias through elimination of unnecessary categorization of the objects in the object space;

4) ease for evolving of the global view;

5) ease in the merging of partial views when using a requirements-first approach to database design;

6) ease of mapping into other models, in some cases using existent computer-aided database accommodation packages;

7) trend among researchers toward smaller logical units in logical database design;

8) availability of a binary associative pilot system.

These arguments are based on an analysis of the structure of the conceptual schema. Mapping considerations which will be important in the implementation of a coexistence architecture also favor a model based on the smaller and semantically more explicit constructs. Therefore, for the modeling of the chemical plant design process a binary associative model is selected.

3.4 The role of the data dictionary in logical database design

A data dictionary is a collection of metadata, i.e. data about data in the enterprise's information system. Under this definition a data dictionary can be used regardless of the degree of sophistication an information system has reached (LOMJ78), and will be useful during design, implementation, operation, and maintenance. There are some problems derived from the varying degree of knowledge available during these four stages in the life-cycle of an information system and

from the characteristics of existing data dictionaries. For a data dictionary to provide useful information and statistics during the operational phase, it has to be tuned to the information system under consideration. If the information system is centered on a database, the data dictionary must be compatible with the particular DBMS being used. At the early stages of database design, however, the specification of a particular DBMS should be avoided. Otherwise artificial constraints will be imposed on the emerging global view. Therefore, a data dictionary that is DBMS independent is required at the early stages of database design and should be incorporated into the database design methodology.

Existing data dictionaries present various shortcomings that disqualify them in their present versions as effective support tools for logical database design, and particularly in the context of the present research[3] .

Comercially available data dictionaries can be classified as DBMS dependent and DBMS independent. DBMS dependent systems rely on a particular DBMS and run as an

[3]The information on commercial data dictionary/directory systems stems primarily from references (LEOB76, LOMJ77, DBDS76). For DBMS independent systems the validity of these data was confirmed through personal communications (DACD80, BECB80).

application of that system. They are tailored to that DBMS and require an early commitment to it. Typical examples are UCC TEN, IBM DB/DC and TOTAL DICTIONARY. The DBMS independent systems such as LEXICON, DATA MANAGER, and DATA CATALOGUE have an internal data structure that is hierarchical, with LEXICON claiming also network and relational structures, but LEXICON currently is not being actively marketed (BECB80). The philosophy behind existing data dictionaries is also opposed to that underlying the present research. These data dictionaries provide mostly for registration of the information about data items in the database context after definition of a logical schema. This is best illustrated by the data dictionary entities supported, such as fields, records, groups, areas, and files. Emphasis is also placed on automatic identification of data-items from existing code, which leads to confusion between data-objects in the modeling phase and variable names. In some instances accommodation is allowed, via interfaces, of the global view to particular DBMSs, typically IMS, ADABAS, TOTAL, IDMS and SYSTEM 2000. These interfaces are certainly of interest for generation of the logical schema in terms of the DDL of any of those DBMSs, but accommodation should occur from a global view obtained through a step-by-step logical design, the

portion lacking in commercial dictionaries.

An additional constraint to the use of commercial data dictionaries is their hardware dependence on IBM equipment with the rare exceptions being DATA CATALOGUE available on Univac and Digital (only available as Digital internal software and not available to the public) and ICL 2900 running on ICL equipment. This hardware restrictions coupled with the cost (typically around 10,000 US$ for acquisition) and constraints involved in modification of proprietary software for an academic environment fully eliminated commercial data dictionaries from consideration in the present research.

The obvious alternative was the definition and implementation of a data dictionary/database design tool which supports binary associations. The contents of this data dictionary can be mapped into another dictionary once the logical design is completed, in the same form as the global view is accommodated to a particular DBMS.

The following sections describe the role of the data dictionary in the context of a database design methodology, the envisioned content of the data dictionary, and the functions a data dictionary has to support in order to fulfill its purpose within the

database design methodology.

3.4.1 The data dictionary as a part of the database design methodology

The data dictionary in logical database design is a database containing metadata. The use of the data dictionary means layering the database in such a way that the abstract objects in the application database become values or instances of the meta-objects in the data dictionary. In a data dictionary which stores a global view of the database defined in terms of binary associations only three meta-objects would be needed to describe the global view: associations, subjects, and predicates. Each data object in the application database can have many instances, but in the dictionary the data object's name becomes an instance. For example, stream-number can have 200 instances in a given application, but in the data dictionary stream-number is a value of the meta-object "subject".

When modeling a complex environment which requires application of a database design methodology the number of objects and their interrelationships is too large to keep track of in a completely non-automated form. Therefore, the data dictionary, which should

provide a mechanism to record the necessary information about the database design, is intimately tied to the database design methodology. The dictionary can serve as an index to the database and the design decisions taken earlier. Further, if adequate support routines are defined, it can serve as an aid to detect redundant associations, resolve naming conflicts, keep information on functional dependencies, aid during clustering of data-objects into aggregates, present subportions of the database affected by proposed changes, keep information about affected users, and be the receptacle in which the conceptual structure of the database in the form of local and global views can be stored. The data dictionary can be looked upon as the tool necessary to support the actions laid down in the database design methodology.

It is not intended to incorporate a 'cure-all' data dictionary into the database design methodology, since implementation and operation are not part of the logical database design. Instead, a data dictionary facility has to be defined which supports all tasks to be performed during the logical database design sequence and which can interface with later stages in the database design. Being a database itself, it will be defined in terms of CS4, the binary associative DBMS used in this

work for testing of local and global views. As discussed in previous sections, binary associations can be mapped into any model supported by commercially available DBMSs. Therefore, it is safe to assume that a data dictionary defined in terms of binary associations can be mapped into a commercially available dictionary/directory. This mapping should occur during accommodation of the global view to a particular DBMS and the data dictionary should be compatible with that DBMS to support the implementation and operation phases. At that stage it is perfectly acceptable that the data dictionary runs as an application of any of the more restrictive DBMSs.

3.4.2 The content of the data dictionary

The content of the data dictionary necessary for logical design falls into three categories.

1) Design requirements

The requirement specification consists of data object identifiers, transaction descriptions, performance requirements, data volume and transaction rate information, and updating, alerting and security constraints.

2) Description of local and global views

Since it was decided earlier that binary associations were to be used this information consists of all the association, subject, and predicate triplets.

3) Definition of equivalencies and aliases

Since only names of associations, subjects and predicates are stored and operated on, this category represents the necessary information to identify equivalent names or aliases. In addition to aliases, the equivalent data-objects, i.e. pairs with a bidirectional functional dependence, should be identified.

3.4.3 Desirable support functions to be contained in the data dictionary

Some desirable functions within the data dictionary can be envisioned. The intention of these support functions is not an automatic execution of the database design operations, since, whenever semantics are involved, human intervention is advisable. The functions to be contained in the data dictionary are mostly sorting facilities and report generators that present the design information in a form useful to the database designer. This makes the data dictionary an interactive database

design tool.

1) Resolution of homonyms:

Homonyms can come from different partial or local views. The initial task is to check whether two objects in two local views to be merged carry the same name. If so, the data dictionary has to report these common occurrences and the database designer must decide by inspection whether the common name refers to the same object, and if not, one must be altered.

In the case of association names, the data dictionary requires a reporting function for associations with the same name. The same name is allowed if and only if subject and predicate are the same, otherwise an alert has to be issued for the database designer to change the name of one association.

2) Resolution of synonyms:

Resolution of synonyms is more difficult than that of homonyms. One feasible approach is a sort on two of the three components of every triplet of a pair of local views to be merged. The third component can then be examined to decide whether it is used with the same semantics or not in two associations being otherwise the

same. The main problem of this approach is the startup of the synonym resolution process, since an undisciplined naming policy can mask synonyms, especially at the beginning of the resolution process. This problem can be eased by establishing and using a naming discipline while generating the local views. If difficulties with the search on two elements are expected, associations with only one common element could be presented to the designer for interactive analysis and resolution of synonyms. One possible aid for identification of synonyms consists in storing key-words associated with the data-object in the data dictionary, for example, the units or dimensions, the source or an abstraction of the data-object. Synonym resolution based on keywords can also be executed during data-object definition.

3) Elimination of redundancy:

After naming has been standardized, redundant associations can be eliminated. This function is intimately related to the resolution of synonyms. It consists of identifying the associations in which subject and predicate are the same, but the association name is different. This allows the designer either to eliminate one association and replace it by its equivalent in the corresponding procedures or accept both associations

independently from each other.

4) Definition of aliases:

If the naming discipline provides for names that are directly related to the application it is convenient to define a directory of aliases in the DD during name standardization. Such a directory will be useful in the definition of the external views.

5) Extraction of subportions of the database:

This function implies the extraction of all associations which contain a certain data object, either as subject or as predicate and its report to the database designer. The report can be a list of triplets or it can be presented in graphical form. Plotting would imply the spatial distribution of each node and its corresponding arcs, such that maximum clarity is provided. Algorithms, such as those studied for automatic spatial layout of flowsheets (BROM70) with planarity checking and minimization of arc crossings can be helpful. Additional criteria, such as weighting important nodes could be included to increase visibility of the critical portions of the displayed subschema.

6) Aggregation of data-objects:

The clustering of data-objects based on their functional dependencies needs to be supported by the data dictionary. The aggregation procedures should be able to retrieve the necessary information from the data dictionary.

7) Record of requirement specifications and design decisions:

The DD ought to contain a full description of the requirement specification and should document every design decision taken during the logical design process to serve as a guideline during future expansions and as justification if the results are questioned by management.

8) Size estimates for the finished database:

This support function has to derive the desired information from the size of the global view identifying the number of different objects in it and the expected cardinality of some dominant objects.

These are DD facilities which can be identified in advance. The exact definition of each function has to occur in parallel with the detailed specification of the

database design methodology and is presented in a later chapter.

3.5 Requirement specification languages as an alternative to the definition of a data dictionary

An alternative to the use of a data dictionary is the use of requirement specification and analysis techniques. These techniques have proven useful in the design and implementation of large software systems. Typical examples are SOFTECH's SADT (ROSD77a,ROSD77b) and PSL/PSA (TEID77) which was developed as part of the ISDOS project at the University of Michigan. SADT is a predominantly diagrammatic approach to determine the flow of information in large software systems. PSL/PSA (Problem Statement Language/Problem Statement Analyzer) is a package developed for similar purposes. It includes a language which allows machine readable statement of a problem and PSA is essentially a report generator that allows crossreferencing of the information kept in a CODASYL-DBTG type database. Both systems share a top-down approach to problem statement and analysis which is appealing for gradual refinement of a system. The problem in this approach lies in the integration of different subportions if these overlap partially with

existing portions of the system. In the case of databases this translates into the a priori categorization into entities, atributes and relationships, very much in line with the underlying network model. As has been argued before such an early categorization has no obvious benefit but results in problem when integrating local views. In the light of this difference it was decided that, even though many similarities and common functions exist, for a requirements first methodology it is justified to define the corresponding binary associative data dictionary.

CHAPTER 6

SYNTHESIS OF A LOGICAL DATABASE DESIGN METHODOLOGY RESPONSIVE TO ENGINEERING DESIGN REQUIREMENTS

In Chapter 4 desirable features for a logical database design methodology were formulated and Chapter 5 described the means of homogenizing the representation of graphical and non-graphical information. This chapter introduces the methodology which was synthesized adhering to the guidelines established earlier. Section 6.1 represents an overview of the whole methodology, while sections 6.2 through 6.9 will expand on each of the steps. A description was attempted in sections 6.2 through 6.9 of the inputs and outputs for each step and to provide an implementation-independent description of the tasks to be performed. To avoid mixing the database design principles with the implementation of particular design tools, at the end of each section a list of pertinent data dictionary routines is given. A description of the implemented procedures and their mode

of operation can be found in Appendix C. Portions of an example that illustrate the use of the methodology are given in Chapter 7. For reasons of continuity it is suggested that the whole methodology is read once and that the reader refers to the individual sections and Appendix C for greater depth when following each step of the example.

6.1 Description of the methodology

The methodology consists of nine steps as depicted in Table 9. First an environment analysis is performed which has as its main purpose the definition of the data handling need of the enterprise resulting in the selection of a global architecture, both for the database and the required hardware environment.

The next step is the requirements gathering and analysis which has four sections: information requirements, processing requirements, communication requirements, and constraints. Data-objects are defined and a source and a generalization for each are identified. Based on these keywords, synonyms are resolved. With the previously identified data-objects a data element diagram (DED) as introduced by Raver and Hubbard (RAVN77,HUBC78) is drawn and from there the

TABLE 9

OUTLINE OF PROPOSED METHODOLOGY

INPUT	STEP	OUTPUT
data handling problem reports projections about data usage enterprise's priorities, constraints	environment analysis	global database organization global database architecture definition enterprise's hardware environment
users' information needs users' processing needs users' priorities and constraints	requirement gathering homonym synonym resolution	DD-based description of user-specified data objects, information requirements, processing requirements
users' information needs data-object descriptions	modeling local views generating DEDs	DEDs for each application; abstract objects as keywords
DEDs and functional dependencies mapping information between data-objects	analysis of diagrams and mapping information, aggregation	set of binary associations underlying one application aggregates
user requirements set of binary associations	testing of local view	correct, user-callable CS-4 procedures refined set of binary associations
set of binary associations local view set of binary associations previous global view, DD definitions and keywords	merging of local view into global view with homonym and synonym resolution	integrated global view resolved naming
binary associative global view user callable CS-4 procedures	testing of global view	refined global view refined CS-4 procedures
binary associative global view mapping information from DD	accommodation of the global view to a commercial DBMS	logical schema in DDL of selected DBMS

functional dependencies are identified as far as possible. Based on the known functional dependencies an aggregation algorithm (MIT76a,MIT76b) is suggested for identification of aggregates, which serve for better visualization and as the basis for mapping to 3NF relations.

Implying to binary associations is done to define logical access paths and to be able to test and store the relationships that exist among the data-objects in the data dictionary. The binary associations previously defined are tested by execution of procedures which represent the user's information requirements and rely strictly on the defined binary associations. The test-database and the procedures are implemented in CS-4 [5] and its metalanguage . When defining the testing procedures the preferred logical access paths are specified. The binary associations that are used in any of the logical access paths are stored, together with their mapping information, and later integrated into the global view once a local view is regarded as correct.

[5] A brief description of the CS-4 system is given in Appendix B. For more details reference JAN79 should be consulted.

Merging local views at the binary associative level eliminates problems associated with different categorization of the data objects in the various user views. The merging process becomes the process of taking the union of sets of binary associations. The problems associated with this process are homonym detection for association names and resolution of synonyms among data-objects, which is also a major problem in any other approach to the merging operation. Synonym resolution is handled with the help of the keywords for each data-object stored in the data dictionary.

Since merging may require the resolution of homonyms and synonyms and the resulting elimination or modification of previously defined associations, each user view has to be tested against the integrated binary associative database. The set of all the binary associations obtained after merging of the various local views is the global view of the database.

The global view can be mapped into a logical schema in terms of a particular DBMS using as input to automated accommodation packages the set of binary associations and the mapping information obtained earlier and which was kept in the data dictionary. This information is enough to interface with packages like

IBM's DBDA while accommodation packages designed for relational systems can, in addition, build on the definition of aggregates, the keys already determined and the information on functional dependencies stored in the data dictionary. An outline of how the data dictionary information could be used beyond the strictly logical design is given in Appendix E.

6.2 Environment analysis

The environment analysis is the step in which the user community analyses the data handling requirements and decides on a certain type of data handling system. The inputs to this step are data handling problem reports, derived from the inadequacy of the present system, long-range projections about data usage, the enterprise's priorities and constraints, and the anticipated hardware environment.

A set of questions which ought to be asked (and answered) at this stage includes:

* What have been the main problems in the current system?
* Can these problems be solved effectively by a database system?
* Who are the potential users of the database system?

* What is the expected update/retrieval ratio?
* How fast has an update to become active?
* What is the anticipated volume of data?
* Will the database grow, stay constant, or fluctuate in size?
* Can data be discarded periodically?
* How many users will access the database simultaneously?
* What is the propagation effect of updates?
* Can layering or segmentation of the database alleviate size/concurrency problems?
* Is extensive processing done with retrieved data?
* Should processing be done within the database system or externally?
* Can the present hardware environment support a database system?
* What is the projected hardware environment?
* What would the consequences of a crash be?
* What are the advantages of one mainframe vs. a network?
* Is a distributed system envisioned for the future?

The results of this analysis lead to the problem definition and selection of an overall architecture described in the first three chapters. The expected results are a global database organization, such as the

hierarchization into global, project-wide database and local, task-oriented working databases proposed for the chemical process plant design environment. It should also lead to the specification of a desirable hardware configuration, such as a network of mini-computers and their most appropriate interconnection or one megacomputer on which all processing is done (LEEM78b). Detailed estimates of necessary processing capabilities can only be made after the users' processing and information requirements have been analysed but the global database organization will be influenced by the selected hardware configuration.

6.3 Requirement gathering and analysis

Once a global database configuration has been agreed upon and the enterprise's priorities and constraints have been formulated and considered, it is necessary to evaluate the individual users' requirements. The term 'user' is employed liberally and may mean an individual requesting information interactively as well as a program retrieving data to be used in a calculation routine. This information will determine the structure of the conceptual, external and internal schemata and

should, therefore, be as complete and precise as possible.

When trying to determine the users' requirements for a database system four basic categories of specifications can be recognized:

* information requirements;
* timing requirements;
* format and media requirements;
* constraints.

6.3.1 Information requirements

Information requirements are basically a specification of what the user expects to be in the database and what he expects to extract from it. The primary source for this information is an analysis of existing application routines and non-computerized activities. In the case of a design environment like the one under consideration, records are kept of all pertinent information, computerized or not, on predefined forms which are filled out by the designers and serve, together with the various kinds of drawings, as the principal means of communication. Such document blanks are an ideal source for identification of the data-objects as seen by the user. Existing documents are

a good starting point, but an effort should be made to identify new applications of interest to the enterprise.

Data elements should be atomic, i.e. no hidden subelements should be contained in them. In order to standardize the information provided by each user, the data dictionary is designed to help the user by prompting him for the necessary information. To set the basis for easier homonym and synonym resolution a naming policy is proposed. Adherence to this naming discipline increases the probability that the same object will receive the same name even when given by different users or database designers. The naming discipline consists of such simple things as using the same graphem for the same concep'T', for example, 'T' for every temperature rather than once using 'T', another time 'TEMP' and a third occasion 'TEMPERATURE', or 'DEL' for every increment or difference rather than using 'D', 'DEL', 'DELTA' or 'DIFF' indistinctly. The particular naming discipline is environment-dependent and needs to be defined such as to be compatible with application jargon and company policy. A possible naming discipline for the plant design environment covering the terminology appearing in the example is proposed in Appendix D.

The parameters to be specified during the object definition are, in addition to the name: the application in which the data-object is used; the source, i.e. definer or provider of those particular data-values; an abstract object or generalization of the data-object; and a description of the data-object. Source and generalization serve as keywords for synonym resolution. The selection of the keywords is important for effective synonym detection.

Source is used as a keyword based on the assumption that only one user should be ultimately responsible for the values of a data-object. Although this keyword is not unique, since one user can be responsible for many data-objects, it helps to reduce the number of potential synonyms.

The abstract-object is a generalization in the sense introduced by Smith and Smith. The differences are ignored in favor of a dominating common characteristic, but only one level in the generalization hierarchy is provided. For example, Cp of the liquid phase at T and P, Cp of the vapor at T and P, and Cp of the vapor at STP are all heat capacities. Generalization was chosen because of its applicability to all kinds of data. In some work-space databases, the dimensions can be used

instead. In that case it would be important to use dimensions rather than units to avoid conflicts with multiple unit-systems or within a system assigning, for example, 'inches' or 'feet' for the dimension 'length'.

When defining a new data-object it is necessary to guarantee that the name, as the primary identifier, is unique. Therefore, before a new data-object can be inserted into the data dictionary it is checked for homonyms already existing in the dictionary. It is equally important, and more difficult, to detect any synonyms. This is done by comparison of the keywords described above. If a match of keywords occurs the data-objects with their descriptions are displayed. This process reduces the search space considerably and allows the database designer to inspect the synonym candidates and decide whether they are true synonyms or not. If they are, the new name is stored as an alias of the existing name in an alias table.

The description is a brief definition of the data-object to be used in case of doubt about the meaning of the name and is particularly useful for synonym recognition.

The cardinality is an estimate of the anticipated maximum number of instances of that data-object that will be held in the database. This information is useful for size estimates and during accommodation. The specification of the application a data-object is defined in allows easy retrieval of dictionary information by application rather than specification of individual data-objects. Application and local view are used in the same sense throughout this chapter. However, application has a wider meaning. For example, when referring to the material balance application, all data-objects intervening in a material balance are included. Several local views, such as retrieval of the whole information associated with a material balance, complete information about one stream, or only flowrate, temperature, and pressure of a stream may be of interest. Local views, therefore, can be subsets of the information handled in one application.

6.3.2 Timing requirements

Timing requirements describe the users' processing needs and indicate when and how fast the information is needed and they have to be specified in two areas: the timing relative to project start-up, and

the timing relative to query statement. The first category is necessary to estimate the time of release for each set of data from the work-spaces to the project-wide database and to establish estimates of average and peak-loads relative to a project's life-cycle. Timing relative to query statement is critical, since response-time is a limiting factor in very large databases. Interactive processing, in particular, depends on reasonably short response-times and the users have to indicate what the maximum permissible delays are. This information is helpful in assigning the more efficient access-paths to those queries with stringent response-time constraints. Both kinds of timing information together serve to estimate required hardware capabilities.

Timing information is only used in the accommodation phase since an approach to logical database design based on information requirements is advocated. In the accommodation step, however, availability of detailed timing information is critical in the selection of keys and access paths. The actual response time can only be estimated if the workload is considered simultaneously, therefore, each user is asked to provide estimates on the frequency with which a transaction is

executed. Timing requirements are specified only relative to a local view, i.e. a user specified operation which may consist of many physical or atomic transactions, such as retrieval, insertion, deletion, and update. Even though timing requirements are not considered during most stages of the database design discussed here, it is convenient to gather them at the same time as the information requirements. Since users are prone to exaggerate their requirements, an inexpensive and convenient way to minimize unreasonable requests is by keeping a record of the individuals who specified a certain requirement in the form of a definer's code for each application. This also creates a path to the data-objects and their definers so they can be consulted should any conflict arise.

6.3.3 Format and media requirements

These requirements are fairly easy to specify and they comprise for each logical transaction the specification of display format, such as report, single query response or graphical output. The medium can either be line printer, alphanumeric CRT, graphics CRT, and high speed or high quality plotter. It is convenient to identify a primary and a secondary output device for

hardware estimates.

6.3.4 Constraints

Constraints are an ill-defined aspect which can range from consistency and integrity constraints, to authorization schemes and proliferation consequences, to security and hardware limitations.

Security is not a major issue for the bulk of engineering data, since access to the whole database will be limited to personnel involved in a certain project and little harm will be done if access is limited to retrieval with controlled access only for update. However, proprietary information, such as certain process conditions, catalyst data, or some financial data should not be available to everybody. One possible solution is based on the characteristics of the design environment. Designers seldom require isolated data, they are usually interested in a report which places the requested data into context. This suggests a security scheme based on existing 'mailing lists'. These mailing lists are documents which indicate who should get a copy of which document. Passwords can, therefore, be screened at the procedure level rather than the data-object level. The update problem is discussed best in the context of the

update proliferation consequences. Updates are held as part of the database being data-objects themselves. The previously mentioned mailing lists serve again as a guideline for implementation of the authorization scheme. An update is only in effect after approval or acknowledgement by all those affected. Therefore, for each individual application which disseminates data a mailing list should be kept in the data dictionary.

Consistency checks are necessary before updates are accepted. Currently most checks are embedded in the form of input verification in the application programs. These checks are needed and should be preserved, but in a database environment they would only verify data after they have been stored in the database. Consistency checks are required in the database to ascertain that data which are redundantly defined are not contradictory. For example, the liquid fraction of a stream can be calculated as (1 - vapor fraction), and vice versa. From requirement gathering the designer gets both data-objects. It is part of the requirements analysis to recognize this reduncancy and record the appropriate derivation rule. In the accommodation stage these rules will be valuable in implementing the consistency constraints. If for performance reasons it was chosen to

have a data-object whose value can be derived from other data stored explicitly in the database the derivation rule becomes a consistency constraint. Such a constraint can be used for checks when updating that data-object, but for retrieval the redundant data-object could be used unconstrained. This is particularly useful in an environment with low update/retrieval ratio and whenever deduction requires several database accesses and additional computation. For example, the heat transferred in a heat exchanger can be calculated in different ways: as the enthalpy difference between input and output streams multiplied by the flowrate, or based on the flowrate, the heat capacity and the temperature difference. In either case 3 or 4 database accesses and additional processing are required. Storing the duty of the exchanger explicitly results in obvious performance increases, which are not a major concern at this stage but will be a factor in the accommodation phase.

6.3.5 Data dictionary support functions for requirement gathering and analysis

The data dictionary functions provided for support of the requirement gathering operation are:

for initial requirement definition OBJECTDEF, ALIASDEF, TIMINGDEF, DISTRIBDEF, and CONSTRDEF;

for requirement retrieval and reporting OBJECTRET, ALIASRET, TIMINGRET, DISTRIBRET, and CONSTRRET;

for requirement modification OBJECTUPDATE, TIMINGUPDATE, DISTRIBUPDATE, and CONSTRUPDATE.

6.4 Modeling of local views

Once the different applications have been identified and their requirements have been specified and stored in the data dictionary it is time to model each local view. Modeling the local views is, in fact, one of the most difficult tasks in the logical database design, since it involves establishing the relationships among the data-objects of one local view.

The modeling of a local view consists of five subtasks:

* generation of a data element diagram;
* definition of auxiliary data-objects (and those data-objects that were missed before);

* identification of functional dependencies and derivation rules;
* resolution of identities and identification of candidates for primary and secondary keys;
* aggregation of data-objects and recording of keys.

6.4.1 Generation of data element diagrams

The first task is the generation of a diagram connecting the data-objects in graphical form according to the relationships of interets for a particular local view. Each directed arc between the nodes representing the data-objects carries information about the type of association the arc represents. An arc is drawn for each association (two for one relationship) and next to each arc a '1' or 'M' is placed. This allows to identify if the mapping is 1:1, 1:M or M:N. This diagram was introduced by Raver and Hubbard and is used in DBDA (RAVN77). The conditional case, which is considered in the original formulation of DEDs, was not deemed necessary since it is reduced in the accommodation to any of the above.

However, it was recognized that in 1:1 correspondences there is often one data-object involved,

which from a logical point of view should uniquely identify another data-object, but for practical reasons is not useful as primary key. Such cases are marked as F (for fuzzy). Any data-object that takes long, descriptive values falls into this category. Examples are equipment number (EQID) and equipment service (EQSERV). EQID can only take one perfectly defined value for each instance and that value is a unique identifier under all circumstances. EQSERV is descriptive. Even though conceptually there is only one equipment that performs a certain service, the descriptive nature of EQSERV makes it ambiguous. 'Deethanizer reboiler' indicates the service correctly in some cases, but should we have an interreboiler, the previous description would not be unique and we would have to use, for example, 'deethanizer bottoms reboiler'. Knowing about the fuzziness of a correspondence allows the designer to exercise the necessary caution when selecting keys.

6.4.2 Definition of auxiliary data-objects

The generation of the DED and the specification of the mapping information reveals any data-object which depends on more than one data-object to be uniquely determined. Whenever such a situation arises an

auxiliary data-object is defined. The role of this auxiliary data-object is to permit representation of concatenated keys in a binary associative model. Each auxiliary data-object is stored in the data dictionary and integrated into the list of data-objects associated with an application. The objects which define the auxiliary are also stored explicitly in the data dictionary.

The utility of the auxiliary data-objects will be seen when identifying functional dependencies. A functional dependence from an auxiliary data-object to another data-object can be treated in the same form as any other simple functional dependence between two data-objects. The auxiliary data-object can be used in algorithms that generate single entity classes, such as Mijares' algorithm, like any other data-object. Their definition can later be substituted, thus allowing the generation of relations with compounded keys.

6.4.3 Specification of functional dependencies

In order to be able to use an aggregation algorithm the functional dependencies that exist in the local view being modeled have to be identified. Should any non-transitive dependence be missed when using

Mijares and Peeble's algorithm, a larger, suboptimal number of aggregates would be the result.

The functional dependencies are identified from the DED and stored in the data dictionary. The identification of functional dependencies is done by inspection based on the mapping characteristics previously marked on the DED and are then stored in the data dictionary.

Functional dependencies can be grouped into those needed for unique identification of a data-object in the data-base, and those required for calculation of a data-object's value. An example can illustrate this distinction best. During process simulation the enthalpy of a stream is calculated every time it is required from the stream's temperature, pressure, and composition. It is necessary to recalculate the enthalpy every time one or more of those variables change. In the global database, however, it is preferable to consider the enthalpy as a property associated with a stream. Knowledge of the stream identifier should be enough for identification of the streams enthalpy, rather than specification of values for composition, temperature and pressure. Carrying the dependence of H on T, P, and composition along with the dependence on STRMID will

result in an undesirable aggregation. Composition has to be expressed through all the components present and their molefractions, the molefractions depending on COMPONENT and STRMID; T and P will be identifiable by STRMID alone. Inference of the value for enthalpy based on the data of composition, temperature and pressure is possible, but in many cases it may be rather difficult to carry out, since physical property calculations are the crux of any simulator.

During database design one should specify both types of functional dependencies. The identifying functional dependencies will be stored as functional dependencies and used for aggregation, those dependencies representing a complex inference rule will be stored as constraints. It will be assumed henceforth that functional dependence refers strictly to an identifying functional dependence.

6.4.4 Resolution of identities

Identities have to be resolved before entering the aggregation algorithm. An identity is recognizable since there exists a simple functional dependence from A to B and another from B to A. In many cases identities will be spotted in the DED. However, if another local

view presents the same identity, it may not be clear which data-object should act as the primary key. Before inserting a new functional dependency the data dictionary checks if the inverse dependency has not already been defined. Whenever an identity is detected one data-object is singled out as the preferred key while the other is stored as a candidate for secondary indexing.

The difference between aliases and identities should be noted. Two aliases take actually the same value and are fully interchangeable. The data-objects in an identity depend functionally on each other but take different values and represent two different objects or concepts.

6.4.5 Aggregation of data-objects

It is proposed that the aggregation algorithm operates against the data dictionary. This has the advantage of minimizing the work needed to set up the input information. Specification of a local view (or several if it is suspected that overlap may occur) will result in retrieval of all the data-objects in the specified local views together with their functional dependencies. They are used to set up a matrix in which each row and each column correspond to one data-object

with a '1' placed at the intersection whenever a functional dependency exists between two data-objects. Since Mijares' algorithm requires strictly non-transitive dependencies as input, a test has to be performed for non-transitivity in the functional dependencies specified among the data-objects to be aggregated. As described in the review of Mijares and Peebles' methodology the algorithm consists in rearranging the rows and columns according to a decreasing number of 'ones' in each row and selecting the minimal set of rows that covers every other row as the set of single entity classes. Since aggregates obtained through this algorithm all depend functionally on one data-object, this data-object is proposed as key. Through the introduction at an earlier stage of the auxiliary data-objects, compounded keys also can be handled in this form. Keys and their aggregates are stored explicitly. Data-objects that were identified as being part of an identity will be used as candidates for secondary keys. Under the constraints imposed in this research, particularly the definition of auxiliary data-objects which is necessary when dealing with binary associations and provided a test for non-transitivity is performed, Mijares' algorithm performs adequately. As an alternative algorithm the one proposed by Bernstein (BERP77) could be used.

6.4.6 Practical considerations in view modeling

It is necessary to recognize that sometimes more than one iteration of the view-modeling step may be necessary as other local views are modeled. It can happen that the modeling approach which appears best in the context of one local view results impractical or totally inappropriate in another. As the design of the database progresses and the designer gains some insight into other applications, remodeling of a local view may be convenient. An example of such a situation occurred during the present research. When modeling heat exchangers which have a fixed number of input and output streams, it appeared convenient to define four data-objects denoting the cold and hot input and output streams. Having the data-objects STRMCOLDIN, STRMCOLDOUT, STRMHOTIN, STRMHOTOUT only the equipment identifier is necessary as a key to define uniquely the stream which serves as hot or cold input or output. When attempting to model an equipment with variable number of streams, such as a distillation column, the previous approach would be excessively rigid, since it would have required the definition of a data-object for each side-stream or multiple feed. Instead, introduction of a more abstract data-object, STRMROLE, which together with

the equipment identifier forms a compounded key that uniquely specifies the role of the stream, is preferred. Since the more general option has to be used for some types of equipment it is convenient to standardize and remodel the heat-exchanger local view.

Another trade-off in modeling becomes apparent in the previous example. One can either model the object system or parts thereof in terms of a larger number of narrowly defined data-objects or one can reduce the number of data-objects by using more abstract and, therefore, less specific data-objects. Reasons in favor of the latter approach are a reduced number of data-objects with the semantics being carried exclusively in the association names, and more generality. These are the conceptually more appealing motives. Of a more practical nature are the reasons in favor of a larger number of data-objects. As can be seen from the previous example, substituting data-objects by a more abstract one, causes those data-objects to disappear as data-objects and emerge as possible instances, i.e. they are part of the domain over which the more abstract object is defined. At the same time it causes the key to become more complex, in this example the key had to be expanded from EQID to (EQID,STRMROLE).

Among the practical problems arising from too general a definition of data-objects is the identification of sources. For example, the data-object PRESSURE can have many sources: in the role of 'operating pressure' it is defined by process engineering, as 'design pressure' it is defined by the group designing that particular equipment. If the source is to have any validity in update protection it has to be unique. In addition, the source, when it can be defined uniquely, is a valuable keyword for reduction of the search-space during synonym resolution. Finally, mapping into any model that has no link or association naming capability will require a breakdown into more specific data-objects.

The compromise attempted in this research consists in defining data-objects that are specific enough to have a unique source, but at the same time the higher level abstraction, which is carried in the data dictionary is provided as an aid to the designer. Also, whenever new insight showed that generality and flexibility were jeopardized by using too specific data-objects or whenever different local views required a different level of abstraction, the more abstract approach was chosen.

6.4.7 Data dictionary support functions for local view modeling

The data dictionary defined for this research provides the procedures FDDEF, FDRET and FDUPDATE at this stage. FDDEF assists the designer in storing functional dependencies among all data-objects associated with an application, while FDRET provides the means for retrieving this information. FDUPDATE can be used for modification of functional dependencies or unassisted insertion of individual functional dependencies. The routines OBJECTDEF, AUXDEF and AUXRET serve for the definition and retrieval of auxiliary data-objects. AGGREGATOR is used to perform aggregation while AGGREGDEF is used to store the aggregate a data-object belongs to and AGGREGRET is used to retrieve that information. KEYDEF and KEYRET are used for storage and retrieval, respectively, of keys and aggregates for which they are the key. IDENTDEF, IDENTRET, and IDENTUPDATE are procedures used for storage, retrieval and update of identities.

6.5 Mapping of local views to binary associations

Mapping to binary associations consists in expressing the associations registered on the DED in form

of triplets, each triplet having an association name, a subject or from-node, and a predicate or to-node. Two basically different alternatives exist for this step:

* mapping of every association identifiable on the DED to processable associations; or

* specification and storage of those binary associations belonging to preferred logical access paths.

At first sight both alternatives have merits. The first approach appears to be more general. It does not limit the design at an early stage since all possible associations are available for the accommodation step. To illustrate this it is assumed that two alternate paths exist to obtain an instance of a data-object in one local view, for example A-B-C-D and A-E-D. For this local view A-E-D may be specified as the preferred logical access path since this implies traversal of fewer links. During implementation, however, a grouping of A,B,C,D may emerge as the more efficient since another application which is used more frequently requires A,B,C,and D but not E, therefore causing the path A-B-C-D to be preferred. This can only be done if the binary association C-D is given to the accommodation package. The distinct disadvantage is the overhead one incurs when keeping 'superfluous'

associations in the data dictionary, especially when one considers the merging operation. Each additional association in the data dictionary - represents six retrievals and the subsequent operations with larger sets for every association added to the global view after this point and 6N additional retrievals for the associations already in the global view at that point.

The advantage of defining logical access paths and working with subsets of binary associations is the reduced overhead during merging but at the expense of some generality in the final accommodation stage. However, an analysis shows that this loss of generality is relative. If an alternate path should be more convenient in a different local view, the most likely reason why in the accommodation step another path is preferable, the corresponding binary associations would be present in the global view since they were introduced through the second local view. In the previous example the association C-D would be present since it is introduced through the second application. In addition, putting all possible associations into the data dictionary means either more extensive testing or the presence of non-tested paths, both undesirable consequences.

Even though the first alternative superficially is appealing because of its apparent generality, the second option has several practical merits. If one parts from the assumption that for a local view a convenient logical access path can be identified and selected over a less favorable, then the global view should contain a collection of all the binary associations involved in the locally most convenient paths and those associations belonging to suboptimal paths should not be in the global view, and should, therefore, be ignored in the logical schema.

Based on these assumptions and the conclusions drawn therefrom, input of binary associations into the data dictionary is delayed until merging and only should occur after testing of local views and the definition of the most favorable logical access paths.

6.6 Testing of local views

The testing of local views consists of defining CS-4 procedures which can be invoked by the user and serve to test if the necessary binary associations were defined and if the test values are stored and retrieved correctly. Definition of the testing procedures involves the definition of logical accesss paths. A logical

access path is the sequence of links (i.e. binary associations) that have to be traversed in order to obtain the desired values. As described in the previous section, only these binary associations will be included in the global view. Therefore, the accommodated database will be based on the associations specified in the testing procedures and these are going to serve as a blueprint for the definition of the external views. Testing of local views permits checking of the mapping information identified on the DED and checking of auxiliary data-objects to see if the desired results are actually obtained. Any flaw that is discovered will return the design to the modeling step.

The test procedures operate on a database which contains the minimum of data necessary to check the procedures. Even though data could ex-profeso be inserted for testing of the retrieval procedures it is better practice to use the actual insertion routines for this task and let the retrieval procedures operate strictly on the data created by the insertion routines so that a cross-check can be made. This proved particularly useful for identification of faulty search templates resulting from inadequate logical access paths since error messages were obtained every time a search with a

null value in the template was attempted. This implies that testing has to occur in the same sequence as the actual database would be created and used by the final users. It should be noted here that the design of a process plant is actually performed top-down. The database design itself follows these levels, at each level, however, a requirements first approach to database design is employed.

The result of this step are user-callable procedures which can also be used for the testing of the global view. Experience has shown that the definition of the test routines usually leads to the refinement of the set of binary associations defined to model a local view. Testing by the users may also result in a refinement of the requirements.

6.7 The merging of partial views

The merging of one local view to another or to the global view defined so far is the process of taking the union of sets of binary associations. The basic task is to identify if an association present in one set already exists in the other set. If it does, it is rejected, otherwise it is incorporated.

No fully automated approach is feasible for this task since recognition of the semantics involved in the names is required. Therefore, a computer aided approach is taken in which the trivial cases are eliminated automatically and the database designer is presented with a much reduced set of potentially conflicting associations to take the appropriate action.

The strategy employed starts with the identification of associations with the same name. The designer can decide whether the two conflicting associations are actually the same or not. In case they are equivalent nothing is added to the existing global view.

In the case of a match in association names but mismatch in any other portion of the two associations under consideration it is necessary to analyse the non-matching portions to see if they are synonyms. The keywords stored in the data dictionary are a good discriminating tool and a match in the keywords is taken as an indication that both names are potential synonyms. The final decision whether the non-matching portions are synonyms or not rests with the database designer. Depending on that he will either have to change the association name if they are not synonyms or define the

alias and drop the new association.

Another case to be considered is the analysis of mismatches in the association name, which automatically occurs for every new association. Different association names do not guarantee that both associations are in fact different. The possibility exists that either both subjects, both predicates, or both subjects and predicates are the same. In the latter case most likely the association name has to be changed, a modification that cascades into the testing procedures. In case only a partial match exists (for example, both subjects are the same but the predicates are different) most likely the two associations under consideration are different but a synonym test has to be made to be certain. Another possibility is that subject and predicate were defined in different order but are actually present with inverted roles in an association. A check for this case can be easily performed.

If neither subject nor predicate matches occur, the association most likely is different from any other that is already in the global view. At the database designer's discretion a synonym test can be made at any stage. In the present research this test was implemented by searching the set of all the data-objects in the data

dictionary for a match in each keyword. The data-objects matching on a given keyword are retrieved and placed in a separate set for each keyword. The intersection of these sets contains the data-objects which are candidates to be synonyms. Displaying these together with the data-object's description that already exists in the data dictionary for each data-object enables the database designer to resolve synonyms. Currently, synonym resolution is done interactively and the whole global view is scanned. As the global view grows this may become impractical and overnight processing and report generation might be adequate. A further option could be the segmentation of the database only testing those associations and data-objects in which conflicts may occur. Segments can be created by extracting first a subset of the data-objects in the data dictionary. This subset could contain only those data-objects which appear in related applications. For example, the data-objects related to financial applications need not be checked if heat transfer applications are added to the global view. It should also be emphasized that a large number of possible values of the keywords, i.e. more specific keywords, will result in smaller sets to be intersected.

6.7.1 Data dictionary support functions for view merging

The routines available for merging of local views are MERGER which in turn calls SYNRES. The procedure DUMPER allows the retrieval of the set of all defined associations, while MINIDUMPER retrieves the associations used in one application.

6.8 Testing of the global view

Testing of the global view serves mainly the purpose of assuring that the information requirements stated by the users can be satisfied with the set of binary associations existing in the global view. The testing consists in execution of the previously defined user-callable procedures and the analysis of their results. This test should reveal any necessary changes due to the elimination or modification of binary associations during merging.

6.9 Accommodation to commercial systems

Accommodation to a commercially available DBMS is highly dependent on the constraints of the DBMS under consideration. Many constraints can be identified as being common to all DBMSs based on one data model, but

even within each group, particular constraints for an individual DBMS exist. Generally speaking, for hierarchical systems some typical constraints are the existence of only one physical parent, and depending on the particular DBMS the existence of a logical parent, the need for dummy segments in logical relations, and the degree of redundancy permitted. An accommodation package for a network DBMS can ignore checks necessary in a hierarchical system to avoid loops, but the handling of named links introduces new difficulties. Relational systems on the other hand will not require any information on loops or who is the parent of a given record. Weight will be placed on correct identification of keys and functional dependencies requiring an early resolution of ambiguities in the set of concepts to be integrated.

The goal of the present research was identified as being the generation of a global view of the database amenable to accommodation to a variety of commercial DBMSs. This means that some features of the proposed database design methodology may be less useful for one accommodation package than for another. For example, aggregates may be less interesting to an accommodation package designed to generate a hierarchical structure

than what they could be to a relational DBMS accommodation package.

Since the goal is a global view which can be accommodated to different DBMSs it was decided to include the necessary information for the different possible accommodation packages even if some packages include features which allow them to duplicate some of the information kept in the data dictionary. For example, the information required by DBDA would essentially be a set of binary associations and the corresponding mapping information. DBDA offers aids for key recognition and detection of transitivity. Keys are recognized as the data elements from which type 1 associations emanate. Therefore, information on keys and aggregates will not be used by DBDA and some will be generated again.

The accommodation to relational database systems will require less information about mapping characteristics of the binary associations and will rely more on functional dependencies and already identified aggregates. Additional information required by relational accommodation packages is the information that allows adequate partitioning of large aggregates into smaller relations conforming to the actual usage pattern. While the global view should not be governed by such

operational aspects they certainly are part of the accommodation step. Therefore, a brief analysis is convenient to identify the information that is required for partitioning in order to provide the necessary dictionary procedures for gathering and storage of those data. That analysis is presented in Appendix E. The necessary data are estimates of the cardinality of data-objects, usage frequencies and response time constraints and estimates on the number of tuples of the global relation used in a particular local view.

Useful for any model is the availability of a 'clean', i.e. better understood, set of data-objects in which a large portion of synonyms and homonyms has been resolved. The set of binary associations, tested against the various local views guarantees higher accuracy of the mapping information. To increase convertibility and avoid unnecessary duplication all the functions which are necessary in the various approaches, such as synonym and homonym detection, key determination, elimination of transitivity, selection of candidates for secondary indexing, definition of constraints needed for implementation of consistency checks and gathering of execution frequency data, should be eliminated from the accommodation packages and done only once during earlier

stages of the logical database design. Future research needs to be concerned with the generation of accommodation packages for a greater variety of widely used DBMSs. These packages should be able to access the meta-data of the data dictionary to generate on demand a logical schema for a particular DBMS.

CHAPTER 7

APPLICATION OF THE PROPOSED METHODOLOGY AND THE DATA DICTIONARY TO A TEST-CASE

In this chapter the use of the methodology and the data dictionary that were described in Chapter 6 is explained. Information that is usually disseminated in the form of a process release was selected to test the methodology and the dictionary. For clarity, one of the smaller local views that were modeled, the creation of an equipment list, is used to perform the various steps in the methodology and to highlight selected aspects. When necessary, portions of other local views are used to illustrate a point. A report of the design result, the global view in binary associations, will be included at the end of this chapter. This global view represents the data that appear in the project initiation, the material and energy balance, the equipment list, the loadsheets for heatexchangers and those for pumps.

7.1. The data-object definition

The local view to be modeled is a list of major equipment comprising equipment identifiers, the indented equipment number if it exists, and a description of the equipment. The indented equipment number serves to identify equipment provided by the vendor of a major piece of equipment, for example, oil coolers and pumps on a compressor. Also given is the number of the issue during which a piece of equipment was first defined and the date of that issue. Additional information found in this local view is the header information identifying the project, customer, and plantsite. Figure 10 shows the interactive definition of some of the data-objects concerning this local view. The data-objects were defined as they were provided by the user.

As can be seen from the above example, the data dictionary detected that the data objects LOCATION, JOBNO and PROJDESCR already exsisted and only registered that they were also used in the equipment list [1]. A typical case of synonym resolution ocurred with the data-object CUSTOMER [2]. During project initialization the term CLIENT was used while CUSTOMER was used for the same element on the equipment list. Based on the keywords 'SOURCE' and 'ABSTRACTION' they were identfied as


```
                    run objectdef

APPLICATION     :eqlist                                              (1)
DATA-OBJECT     :jobno
THIS DATA-OBJECT ALREADY EXISTS WITH THESE PARAMETERS
    SOURCE          PROJECT
    AOBJECT         ID
    DESCRIPTION     PROJECT NUMBER
    CARDINALITY     1
PLEASE CHANGE DATA-OBJECT NAME OR OMIT IT                            (2)
DATA-OBJECT     :customer
SOURCE          :project
AOBJECT         :company
DESCRIPTION     :name of customer
CARDINALITY     :1
POTENTIAL SYNONYMS FOR  CUSTOMER
 CLIENT              NAME OF CLIENT/OWNER OF PROJECT
?s
EXISTING DATA-OBJECT:client
NEW SYNONYM (ALIAS) :customer                                        (1)
DATA-OBJECT     :location
THIS DATA-OBJECT ALREADY EXISTS WITH THESE PARAMETERS
    SOURCE          PROJECT
    AOBJECT         ADDRESS
    DESCRIPTION     PLANTSITE
    CARDINALITY     1
PLEASE CHANGE DATA-OBJECT NAME OR OMIT IT
DATA-OBJECT     :projdescr                                           (1)
THIS DATA-OBJECT ALREADY EXISTS WITH THESES PARAMETERS
    SOURCE          PROJECT
    AOBJECT         DESCR
    DESCRIPTION     NAME/DESCRIPTION OF PROJECT
    CARDINALITY     1
PLEASE CHANGE DATA-OBJECT NAME OR OMIT IT
DATA-OBJECT     :eqid
SOURCE          :process
AOBJECT         :id
DESCRIPTION     :id of class/number of equipment
CARDINALITY     :500
POTENTIAL SYNONYMS FOR  EQID
?s
DATA-OBJECT     :eqidindent
SOURCE          :process
AOBJECT         :id
DESCRIPTION     :indented equipment number
CARDINALITY     :500
POTENTIAL SYNONYMS FOR  EQIDINDENT
 EQID                ID OF CLASS/NUMBER OF EQUIPMENT
?s
DATA-OBJECT     :eqserv
SOURCE          :process
AOBJECT         :descr
DESCRIPTION     :service/name of equipment
CARDINALITY     :500
POTENTIAL SYNONYMS FOR  EQSERV
?s
DATA-OBJECT     :.
APPLICATION     :.
```

Figure 10. Interactive definition of data-objects

synonyms and reported. The data-object CLIENT was chosen as the preferred data-object and CUSTOMER was stored as an alias of CLIENT. Examples of the possible reports that can be generated on data-objects are given in Figures 11 and 12.

7. 2 Definition of timing requirements

An example of a timing and format requirements report, generated by the data dictionary, is presented in Figure 13. The information used to generate it was defined previously in interactive form in response to prompts by the dictionary for each item.

7. 3 Definition of the distribution requirements

An example of the distribution requirements is given in Figure 14 for the heatexchanger loadsheet local view. This information, except for the authorization code, was extracted from existing document distribution information which is part of Pullmann-Kellogg's operating procedure.


```
*run objectret

DO YOU NEED HELP?no
? ?
APPLICATION : eqlist
    DATA-OBJECT:    CLIENT
    DESCRIPTION:            NAME OF CLIENT/OWNER OF PROJECT
    SOURCE:                 PROJECT
    AOBJECT:                COMPANY
    CARDINALITY:            1
    DATA-OBJECT:    EQID
    DESCRIPTION:            ID OF CLASS/NUMBER OF EQUIPMENT
    SOURCE:                 PROJECT
    AOBJECT:                ID
    CARDINALITY:            500
    DATA-OBJECT:    EQIDINDENT
    DESCRIPTION:            INDENTED EQUIPMENT NUMBER
    SOURCE:                 PROJECT
    AOBJECT:                ID
    CARDINALITY:            200
    DATA-OBJECT:    EQLISTIDATE
    DESCRIPTION:            DATE EQLIST WAS ISSUED
    SOURCE:                 PROJECT
    AOBJECT:                DATE
    CARDINALITY:            20
    DATA-OBJECT:    EQLISTINO
    DESCRIPTION:            ISSUE NUMBER OF EQLIST
    SOURCE:                 PROJECT
    AOBJECT:                DOCNO
    CARDINALITY:            20
    DATA-OBJECT:    EQSERV
    DESCRIPTION:            SERVICE/NAME OF MAJOR EQUIPMENT
    SOURCE:                 PROCESS
    AOBJECT:                DESCR
    CARDINALITY:            500
    DATA-OBJECT:    EQSERVINDENT
    DESCRIPTION:            SERVICE OF INDENTED EQUIPMENT
    SOURCE:                 VENDOR
    AOBJECT:                DESCR
    CARDINALITY:            200
    DATA-OBJECT:    JOBNO
    DESCRIPTION:            PROJECT NUMBER
    SOURCE:                 PROJECT
    AOBJECT:                ID
    CARDINALITY:            1
    DATA-OBJECT:    LOCATION
    DESCRIPTION:            PLANTSITE
    SOURCE:                 PROJECT
    AOBJECT:                ADDRESS
    CARDINALITY:            1
    DATA-OBJECT:    PROJDESCR
    DESCRIPTION:            NAME/DESCRIPTION OF PROJECT
    SOURCE:                 PROJECT
    AOBJECT:                DESCR
    CARDINALITY:            1
?
```

Figure 11. Detailed data-object report; shown here are the data-objects for EQLIST after refinement in the view-modeling step.

```
*run objectret
DO YOU NEED HELP?no
? 3
    APPLICATION:    EQLIST                    DATA-OBJECTS:
                                              CLIENT
                                              EQID
                                              EQIDINDENT
                                              EQLISTIDATE
                                              EQLISTINO
                                              EQSERV
                                              EQSERVINDENT
                                              JOBNO
                                              LOCATION
                                              PROJDESCR

?
```

Figure 12. Report of data-objects used by EQLIST


```
   run timingret
DO YOU NEED HELP?no
?a
APPLICATION :eqlist
    APPLICATION     EQLIST
                                        DEFCODE     APB
                                        EXECFREQ    LOW
                                        BEGPROJ     0
                                        ENDPROJ     0.3
                                        MAXRESP     F
                                        FORMAT      SR
                                        OUTDEV1     LPT
                                        OUTDEV2     CRT
?.
```

Figure 13 Timing, format, and media requirements for EQLIST

```
run distribret

DO YOU NEED HELP?yes
TYPE <A> IF YOU WANT MAILING LIST FOR APPLICATION
TYPE <R> FOR LIST OF DOCUMENTS RECEIVED BY SOMEONE
TYPE <.> TO EXIT
?a
APPLICATION: exchload
    APPLICATION                 RECEIVER(S)              COPIES AUTH-LEVEL
  EXCHLOAD

                          PROJECT ENGINEERING MGR.         1        1
                          PROCESS SYSTEMS SECTION          1        1
                          MATERIAL CONTROL SECTION         1        0
                          EXCHANGER ANALYTICAL             1        1
                          EXCHANGER PRODUCTION             1        0
                          PLANT LAYOUT SECTION             1        1
                          PIPING PRODUCTION SECTION        1        0
                          MGR. PIPING MATERIALS DIV.       1        0
APPLICATION:
?.
```

Figure 14. Distribution requirements for heat-exchanger loadsheet

7.4 Definition of auxiliaries

In Figures 15 and 16 the DEDs for the equipment list and for a portion of the material balance local views are given. The equipment list requires no auxiliary data-objects since no data-objects that depend on more than one other data-object are present. This is different in the case of the material and energy balance-Stream and component together determine the mole fraction of that component in the stream. Therefore, it was necessary to introduce an auxiliary data-object, AUXSTRMCOMP, which was defined via AUXDEF as shown in Figure 17. The definition of an auxiliary data-object can be retrieved using AUXRET as shown in Figure 18.

7.5 Definition of FDs

Next the non-trivial FDs are defined. A portion of the definition of FDs for the equipment list application is given in Figure 19.

An interesting point arises during the definition of the FDs for the equipment list. EQID and EQSERV form an identity. Since EQSERV is functionally dependent on EQID and vice-versa; a message is issued to the designer to choose the preferred data-object for use as a primary

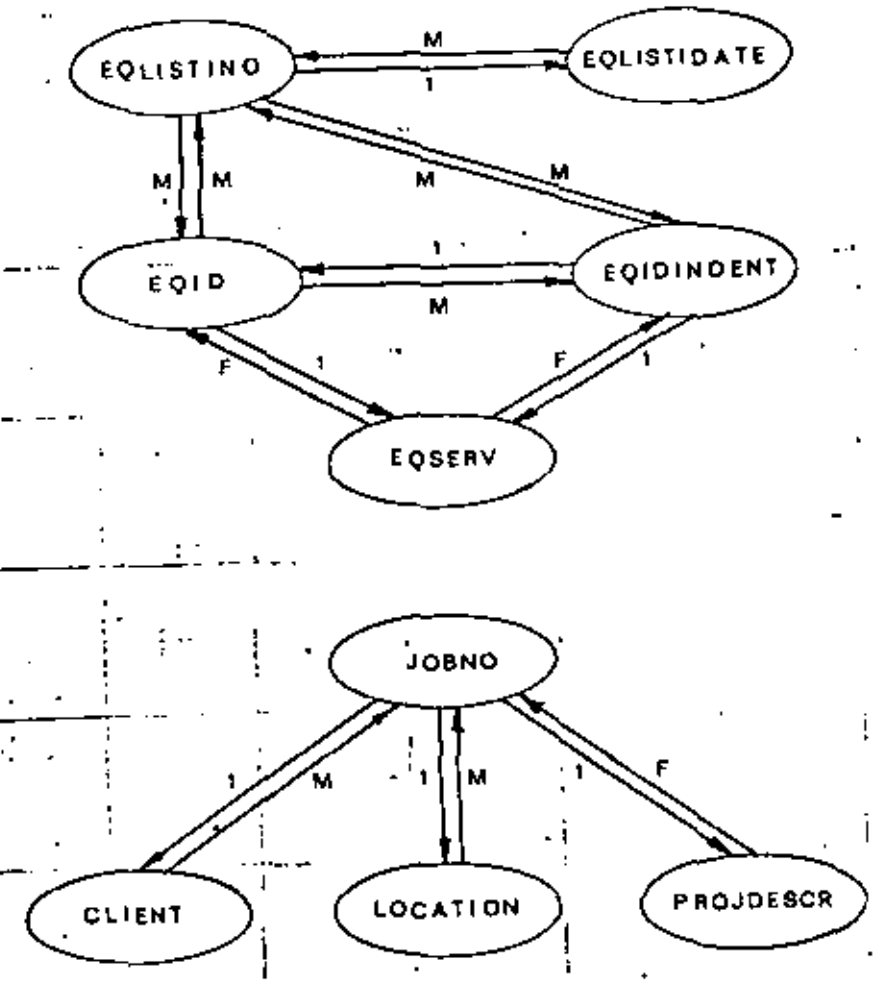


Figure 15. Data Element Diagram (DED) for local view EQLIST

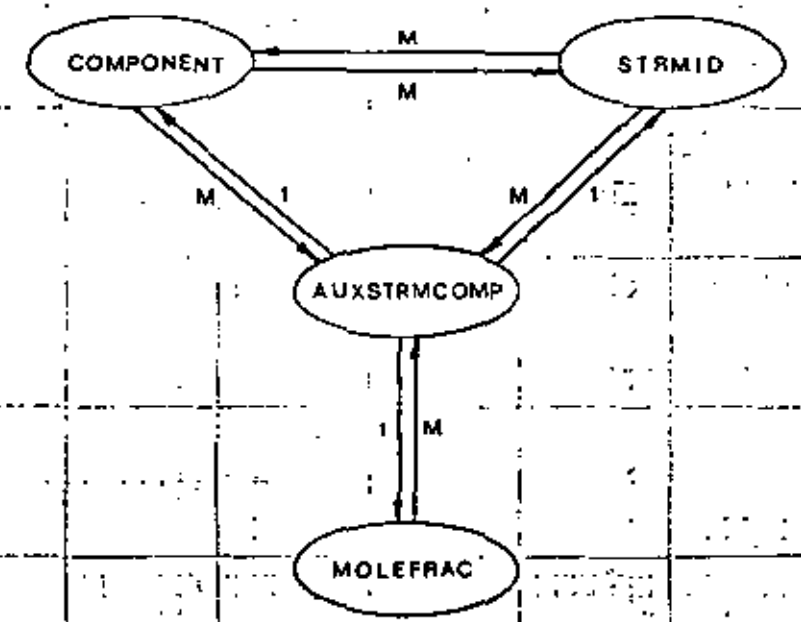


Figure 16. Portion of the Data Element Diagram for the local view MATBAL

```
 run auxdef

AUXILIARY DOBJECT:auxstrmcomp
DEFINING DOBJECT1:strmid
DEFINING DOBJECT2:component
AUXILIARY DOBJECT:.

*
```

Figure 17. Interactive definition of auxiliary data-objects.

```
*run auxret

?h
TYPE <A> FOR ALL AUXILIARIES
TYPE <S> FOR DEFINITION OF A SPECIFIC AUXILIARY
TYPE <.> TO EXIT
LIST OF AUXILIARY D-OBJECTS
AUXASEQ              EQID              ASSIG
AUXEQPROCNOTE        EQID              PROCNOTENO
AUXSTRMCOMP          STRMID            COMPONENT
```

Figure 18. Report on the definition of auxiliary data-objects


```
*run fdupdate

DO YOU NEED HELP?no
?i
INDEPENDENT OBJECT:  eqid
DEPENDENT  OBJECT :  eqserv
?i
INDEPENDENT OBJECT:  eqidindent
DEPENDENT OBJECT  :  eqid
?i
INDEPENDENT OBJECT:  eqserv
DEPENDENT OBJECT  :  eqid
EQID , EQSERV FORM IDENTITY
WHICH SHOULD BE KEPT AS PRIMARY KEY?eqid
?.
```

Figure 19. Definition of funtional dependencies and selection of primary key

```
*run fdret

DO YOU NEED HELP?no
?a
APPLICATION:  eqlist
THE DEFINED NON-TRIVIAL FUNCTIONAL DEPENDENCIES ARE:
    EQID                 EQSERV
    EQIDINDENT           EQID
    EQIDINDENT           EQSERV
    EQLISTINO            EQLISTIDATE
    JOBNO                CLIENT
    JOBNO                LOCATION
    JOBNO                PROJDESCR
?.
```

Figure 20. Summary of functional dependencies that exist in EQLIST based on the data-objects as defined by the user

key. As can be seen in Figure 19, EQID was chosen and EQSERV was placed in a set of candidates for secondary indexing. EQSERV is one of those data-objects that could uniquely identify the equipment number, the issue number and date. However, the descriptive nature and often used abbreviations and acronyms make EQSERV undesirable as the primary key.

A report of the functional dependencies defined in this local view, such as that presented in Figure 20, can be generated for inspection by the database designer.

From the users the data-objects EQID, EQIDINDENT and EQSERV were obtained. This resulted in the diagram of Figure 15. Upon analysis of this configuration it can be seen that a transitive functional dependency exists from EQIDINDENT to EQSERV via EQID. Representation of this situation in a relational model would be difficult. Removal of the transitive dependence would result in the two relations (EQIDINDENT,EQSERV) and (EQIDINDENT,EQSERV). These relations clearly do not represent the correct semantics, since the service of the equipment, EQSERV, would be inaccessible for the major pieces of equipment identified by EQID. Binary associations are capable of handling such a situation simply introducing two different associations, for

example, ('SERV-OF-EQ',EQID,EQSERV) and ('SERV-OF-EQINDENT',EQIDINDENT,EQSERV). Naming these two different associations lead to the understanding that EQSERV was performing two different roles and, therefore, the domain EQSERV was split up. This illustrates that EQSERV is actually a generalization of two data-objects, the service of major equipment, which we shall continue to call EQSERV, and the service description of the indented equipment, EQSERVINDENT. An iteration through previous steps, the data-object and functional dependency definition, results in the definition of a new data-object, the modification of the cardinality of EQSERV and EQID and update of the FD between EQIDINDENT and EQSERV to become EQIDINDENT --> EQSERVINDENT. These changes appear in the report of refined data-objects shown in Figure 11. A revised diagram for EQLIST is shown in Figure 21 and the refined functional dependencies are given in Figure 22.

7.6 Generation of aggregates

The results from the aggregate generation step are given below. Each aggregate includes a proposed key, as can be seen in Figure 23.

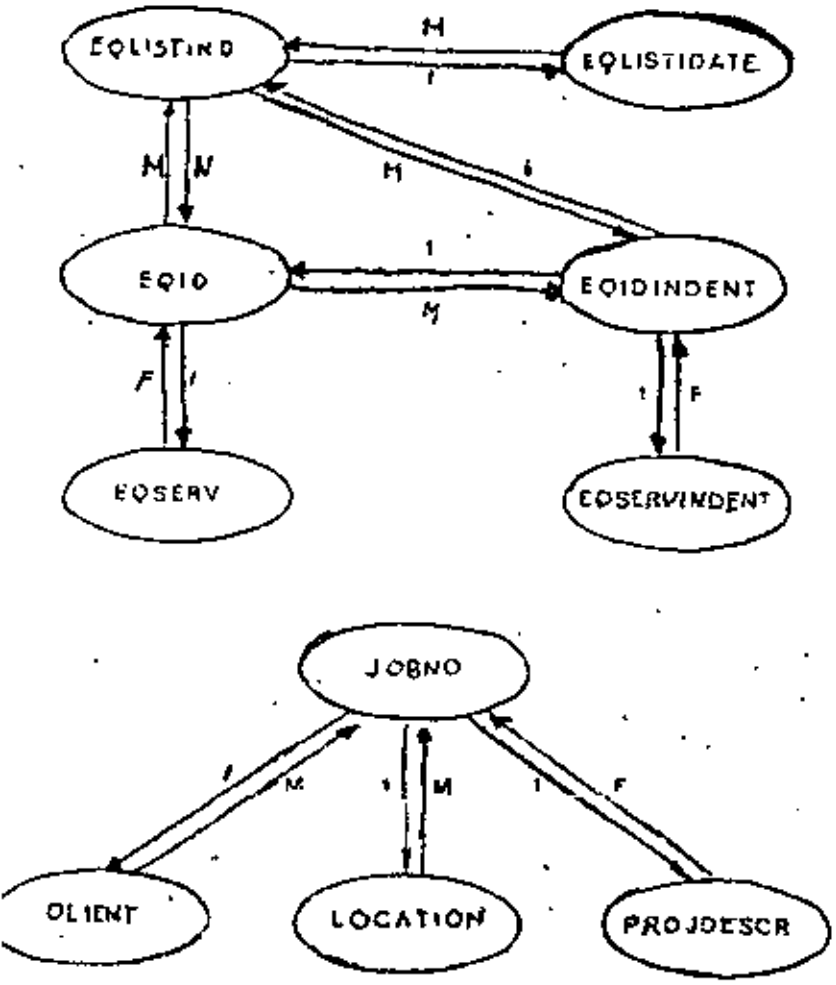


Figure 21 Revised Data Element Diagram for EQLIST


```
*run fdret
DO YOU NEED HELP?no
?a
APPLICATION: eqlist
THE DEFINED NON-TRIVIAL FUNCTIONAL DEPENDENCIES ARE:
    EQID                    EQSERV
    EQIDINDENT              EQID
    EQIDINDENT              EQSERVINDENT
    EQLISTIND               EQLISTIDATE
    JOBNO                   CLIENT
    JOBNO                   LOCATION
    JOBNO                   PROJDESCR
?
```

Figure 22. Refined functional dependencies for EQLIST.

```
*run aggregator

DO YOU NEED HELP?no
APPL:eqlist
?u
DO YOU WANT ANOTHER APPLICATION?no
OBJECTLIST TO AGGREGINT
CLIENT
EQID
EQIDINDENT
EQLISTIDATE
EQLISTINO
EQSERV
EQSERVINDENT
JOBNO
LOCATION
PROJDESCR
NUMBER OF DOBJECTS 10

*****************************
EQLISTINO
EQLISTIDATE

PROPOSED KEY= EQLISTINO
*****************************
EQID
EQSERV

PROPOSED KEY= EQID
*****************************
EQIDINDENT
EQSERVINDENT
EQID

PROPOSED KEY= EQIDINDENT
*****************************
JOBNO
PROJDESCR
LOCATION
CLIENT

PROPOSED KEY= JOBNO
```

Figure 23. Results of the aggregation of data-objects in EQLIST

The previous case shows no auxiliary data object and used as basis for a relational implementation, it can be seen that all attributes depend nontransitively on a single key. The M:M correspondence between EQLISTINO and EQID has to be considered as an auxiliary.

Another situation of interest arises when analysing the output of the single entity class algorithm for the material balance. Here an auxiliary exists and is the key of an aggregate. However substituting this auxiliary by its definition obtained from the data dictionary results in a 3NF relation with a compounded key. This is desirable if accommodation to a relational model is attempted. The definition of auxiliaries is necessary when using a binary model, and this step creates the necessary conditions for Mijares' algorithm to work properly, since it assumes dependence on a single object and cannot handle compounded keys.

7.7 Testing of a local view

A testing procedure was written for the equipment list and was executed with the data for the example described in Section 7.1. In effect two procedures were required, one for input and one for retrieval. Results of the retrieval routine are given in Figure 24.

```
*run eqlistout

ISSUE OF EQUIPMENT LIST:last

                                EQUIPMENT LIST
JOB NO.:   5125                                 ISSUE NO.:   5
CLIENT:    BUCHMANN                             ISSUE DATE:  800404
LOCATION:  AUSTIN,TEX.
PROJECT:   UT OLEFIN PLANT

MAJOR EQUIPMENT INDENTED EQUIPMENT              SERVICE

  M-C-01                                        ETHANE COMPRESSOR
                  D1828                         OIL RESERVOIR
                  D1829                         ACCUM LUBE OIL RUNDOWN TANK
                  E2048                         GLAND SEAL STEAM CONDENSER
                  JE2048                        AIR EJECTOR FOR E2048
  M-D-11                                        DEETHANIZER OVERHEAD DRUM
  M-E-22A                                       DEETHANIZER BOTTOMS REBOILER
  M-E-22B                                       DEETHANIZER BOTTOMS REB. SPARE
  M-E-23                                        DEETHANIZER CONDENSER
  M-E-33A                                       DEETHANIZER INTERREBOILER
  M-E-33B                                       DEETHANIZER INTERREBOILER SPARE
  M-P-03                                        DEETHANIZER REFLUX PUMP
  M-T-03                                        DEETHANIZER
```

Figure 24. Sample equipment list

Testing of local views has proven very useful for detection of misunderstood semantics and refinement of requirements.

7.8 Merging of the local view

The binary associations used in a local view were merged to generate the global view. To illustrate the procedure, three associations from the loadsheet for distillation columns were selected. As indicated in Figure 25, the association marked with a [1] was rejected because subject and predicate were the same as those of an already existing association. Not only were the subject and predicate the same, but also the semantics of the association. The second association [2] had actually the same name as an already existing association. The last [3] was accepted after testing that no association having the same subject-predicate combination, or either the same subject or predicate, is equivalent to it.

7.9 The global view

The database design methodology and its data dictionary were tested through application to a small subportion of the design of a process plant. The local views of project initialization, material and energy

```
*run merger

DO YOU NEED HELP? yes
FIRST PROMPTS SERVE TO CHECK THE ASSOCIATION NAME
OTHER PROMPTS SERVE TO SELCT SYN.RES.,SAVING OR EXIT
OPTIONS AVAILABLE ARE:
SP FOR MATCHING SUBJECT AND PREDICATE IN ASSOCIATION
S  FOR MATCHING SUBJECTS ONLY
P  FOR MATCHING PREDICATES ONLY
K  FOR SYNONYM DETECTION BASED ON KEYWORDS
SV FOR STORAGE OF AN ASSOCIATION
A  FOR STORAGE OF AN ALIAS
.  TO EXIT
-  TO START PROCESS WITH NEW ASSOCIATION
WHAT APPLICATION ARE YOU MERGING?towarload
ASSOCIATION: x-of-strm                                        (1)
SUBJECT     :strmid
PREDICATE   :liqfrac
WHAT OPTION DO YOU WANT?  sp
 X-OF-STRM                STRMID                  LIQFRAC
**************************************************
 LIQFRAC-OF-STRM          STRMID                  LIQFRAC
**************************************************
WHAT OPTION DO YOU WANT? -
ASSOCIATION: vapfrac-of-strm                                  (2)
ASSOCNAME ALREADY EXISTS IN
VAPFRAC-OF-STRM , STRMID , VAPFRAC
MAPPING  1 : M
PLEASE CHANGE ASSOCIATION NAME OR OMIT ASSOCIATION
WHAT OPTION DO YOU WANT? -
ASSOCIATION: feedplate-of-tower                               (3)
SUBJECT     :eqid
PREDICATE   :feedplate
WHAT OPTION DO YOU WANT? sp
 FEEDPLATE-OF-TOWER       EQID                    FEEDPLATE
**************************************************
WHAT OPTION DO YOU WANT? s
 EQINDENT-OF-EQ           EQID                    EQIDINDENT
**************************************************
 SERV-OF-EQ               EQID                    EQSERV
**************************************************
WHAT OPTION DO YOU WANT? sv
MAPPING INFO FROM SUB TO PRED:m
MAPPING INFO FROM PRED TO SUB:m
ASSOCIATION: .
```

Figure 25. Merging of binary associations into global view

balance, generation of the equipment list, generation of loadsheets for heatexchangers, and loadsheets for pumps were integrated to form the binary associative global view shown in Figure 26.

This database structure was tested with data from the deethanizer section of an ethylene plant currently in operation. Testing has allowed to ascertain the correct functioning of the pilot database.

The design of this database also served as feedback to determine the correctness of the information held in the data dictionary and the utility and convenince of its functions. Several new data dictionary procedures were introduced as a result and others were modified.

Design of the pilot database has shown the feasibility of this database design approach and repeated application of the same principle will result in a more comprehensive global view. Suggestions for further work are outlined in Chapter 8.

AUX-ASSIG	AUXASEQ	STRMROLE	1:M
AUX-COMP	AUXSTRMCOMP	COMP	1:M
AUX-CONCMATCORR	AUXMATCORR	CONCMATCORR	1:N
AUX-EQ	AUXASEQ	EQID	1:M
AUX-EQLOADREV	AUXEQLREV	EQID	1:N
AUX-EQPROCNOTE	AUXEQPROCNOTE	EQID	1:M
AUX-LOADADATE	AUXEQLREV	LOADADATE	1:M
AUX-LOADAPPR	AUXEQLREV	LOADAPPR	1:M
AUX-LOADCHDATE	AUXEQLREV	LOADCHDATE	1:M
AUX-LOADCHECKR	AUXEQLREV	LOADCHECKR	1:M
AUX-LOADPDATE	AUXEQLREV	LOADPDATE	1:M
AUX-LOADPREPR	AUXEQLREV	LOADPREPR	1:M
AUX-LOADRDATE	AUXEQLREV	LOADRDATE	1:M
AUX-LOADREVDESCR	AUXEQLREV	LOADREVDESCR	1:M
AUX-LOADREVNO	AUXEQLREV	LOADREVNO	1:M
AUX-LOADREVR	AUXEQLREV	LOADREVR	1:M
AUX-MATCORR	AUXMATCORR	MATCORR	1:M
AUX-MOLEFRAC	AUXSTRMCOMP	MOLEFRAC	1:M
AUX-NOTENO	AUXEQPROCNOTE	PROCNOTENO	1:M
AUX-PROCNOTE	AUXEQPROCNOTE	PROCNOTE	1:M
AUX-STRM	AUXASEQ	STRMID	1:M
AUX-STRMID	AUXSTRMCOMP	STRMID	1:M
AUX-STRMIDMATCORR	AUXMATCORR	STRMID	1:M
COMP-OF-STRM	STRMID	COMPONENT	M:M
CPLIQTP-OF-STRM	STRMID	CPLIQTP	1:M
CPVAPTP-OF-STRM	STRMID	CPVAPTP	1:M
DENSLIQSTP-OF-STRM	STRMID	DENSLIQSTP	1:M
DENSLIQTP-OF-STRM	STRMID	DENSLIQTP	1:M
DENSVAPSTP-OF-STRM	STRMID	DENSVAPSTP	1:M
DENSVAPTP-OF-STRM	STRMID	DENSVAPTP	1:M
ENTHALPY-OF-STRM	STRMID	ENTHALPY	1:M
EQ-ON-EQLIST	EQLISTINO	EQID	M:M

Figure 26. Binary associative global view (continued)

EQINDENT-OF-EQ	EQID	EQIDINDENT	M:1
EQLIST-ISSUED-ON	EQLISTINO	EQLISTIDATE	1:M
FLOW-OF-STRM	STRMID	FLOW	1:M
FLUID-OF-STRM	STRMID	FLUID	1:M
HEAD-OF-PUMP	EQID	PUMPHEAD	1:M
KVALUE-OF-STRM	STRMID	KVAL	1:M
LATENTH-OF-STRM	STRMID	LATENTH	1:M
LIQFLOW-OF-STRM	STRMID	LIQFLOW	1:M
LIQFRAC-OF-STRM	STRMID	LIQFRAC	1:M
LOCATION-OF-PROJ	JOBNO	LOCATION	1:M
MATBAL-PRPD-BY	MATBALINO	MATBALPREP	1:M
MATBAL-PRPD-ON	MATBALINO	MATBALPDATE	1:M
MWLIQ-OF-STRM	STRMID	MWLIQ	1:M
MWVAP-OF-STRM	STRMID	MWVAP	1:M
NAME-OF-PROJ	JOBNO	PROJDESCR	1:M
NONCONDS-OF-STRM	STRMID	NONCONDS	1:M
OVERDESIGN-OF-EQ	EQID	OVERDESIGN	1:M
OWNER-OF-PROJ	JOBNO	CLIENT	1:M
P-OF-STRM	STRMID	STRMP	1:M
PDISCH-OF-PUMP	EQID	PDISCH	1:M
PSUCT-OF-PUMP	EQID	PSUCT	1:M
SERV-OF-EQ	EQID	EQSERV	1:F
SERV-OF-EQINDENT	EQIDINDENT	EQSERVINDENT	1:F
STEAM-OF-STRM	STRMID	STEAM	1:M
T-OF-STRM	STRMID	STRMT	1:M
THERMCONDLIQ-OF-STRM	STRMID	THERMCONDLIQ	1:M
THERMCONDVAP-OF-STRM	STRMID	THERMCONDVAP	1:M
VAPFLOW-OF-STRM	STRMID	VAPFLOW	1:M
VAPFRAC-OF-STRM	STRMID	VAPFRAC	1:M
VAPP-OF-STRM	STRMID	VAPP	1:M
VISCLIQTP-OF-STRM	STRMID	VISCLIQTP	1:M
VISCVAPTP-OF-STRM	STRMID	VISCVAPTP	1:M

Figure 26 (cont.) Binary associative global view

centro de educación continua
división de estudios de posgrado
facultad de ingeniería unam

CASOS PRACTICOS EN EL DISEÑO DE SISTEMAS DE INFORMACION

DISTRIBUTED DATA BASE MANAGEMENT OVERVIEW

DR. ALEJANDRO BUCHMANN

AGOSTO, 1980

Part III: Distributed Data Base Management Overview

These two papers purport to do the same thing: survey activities in the area of distributed data base management. The wide differences between them illustrate the fact that this field today lacks a common framework and common vocabulary.

Rothnie and Goodman survey research and development in the area of distributed DBMS. They assert that no systems of this sort have yet been implemented. Champine, on the other hand, describes six operating implementations. Is one of these papers wrong? No—they are addressing different classes of systems but labeling them with the same name. Rothnie and Goodman discuss *general-purpose* data base management systems (those offering the types of functions described by Chamberlin in Part II) whose implementations are distributed. Champine describes a number of *application-specific* systems in which the supporting data bases are distributed. The simplifications made possible by the application-specific environments have permitted these systems to circumvent the technical hurdles described by Rothnie and Goodman.

The most important element of simplicity that the systems described by Champine have exploited is the restriction to single-record transactions. That is, these systems are capable of retrieving and updating single records but not sets of records. Because of that, the technical problem is reduced to directing data accessing requests to the appropriate site rather than handling requests that deal with data at multiple sites. That simplification makes it possible to implement systems today that perform a useful function for applications that can accommodate its restricted functionality. Champine's paper is the only one in this collection that deals with this type of system. It is valuable reading because it indicates what is possible and, indeed, what is in operation within today's technology.

Rothnie and Goodman describe research and development efforts aimed at achieving complete DBMS functionality in a distributed implementation. The paper classifies these activities according to the principal technical problem areas of the field: synchronizing update transactions, distributed query processing, failure handling, directory management, and data base design. To a substantial degree, this paper constitutes an introduction to Parts IV, V, and VI of this volume by summarizing the topics addressed in those sections. ■

A SURVEY OF RESEARCH AND DEVELOPMENT IN DISTRIBUTED DATABASE MANAGEMENT*

James B. Rothnie
Nathan Goodman

Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139

Distributed database management is a newly developed and rapidly growing sub-field of database management technology. Distributed database management is an attractive approach to solving the data management needs of many organizations because it permits the database system to act conceptually as a centralized system, while physically mirroring the geographic distribution of organizations in today's world.

This paper presents an overview of several key technical problems that must be overcome in developing general purpose distributed database management systems, and surveys current r&d efforts aimed at overcoming these problems.

INTRODUCTION

Advances in the development of computer technology over the past several years have made the concept of distributed database management an attractive one, from managerial, economic, and technical points of view. Distributed database management systems (which we shall refer to hereafter as "distributed DBMSs") permit a collection of data which is relevant to a given enterprise to be managed on a network of geographically dispersed computers. This architecture is illustrated in Figure 1. Distributed data management has been made possible by advances in two areas: (1) the development of relatively inexpensive computing equipment, which has made it feasible for a single organization to operate tens and even hundreds of DBMSs; and (2) the development of "packaged" computer-to-computer communications capabilities which have made it possible to interconnect these large numbers of independent computers straightforwardly, reliably, and at predictable cost.

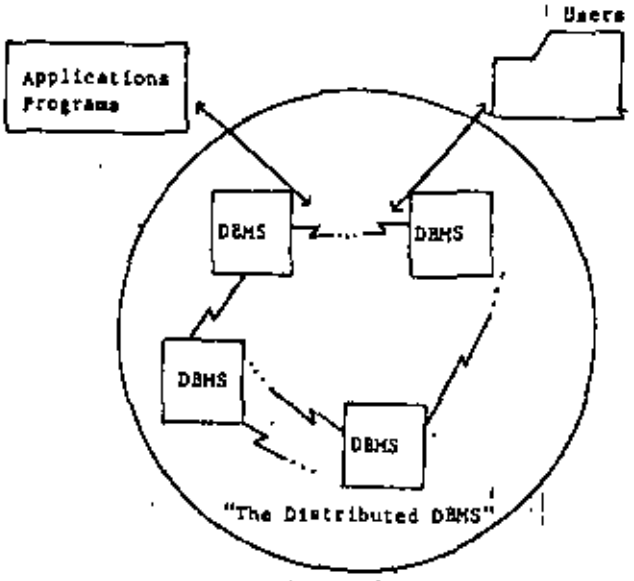


Figure 1

Distributed DBMSs are attractive in applications requiring access to an integrated database from geographically dispersed locations. Applications of this type are common and include activities such as electronic funds transfer in banking systems, inventory control in multi-branch distributors, corporate MIS, military command and control, etc. The distributed DBMS approach offers three major advantages in applications such as these over the alternative of using a conventional centralized DBMS:

1. The distributed DBMS is more reliable than a centralized DBMS because it is constructed from multiple computers located at multiple locations. Consequently the distributed system is not susceptible to total failure when one computer breaks down or one geographic location becomes inaccessible.
2. Since the distributed DBMS exists at multiple locations, it is possible to store portions of the database near to where they are frequently used. As a result faster access to data may be achieved and communication costs reduced.
3. Finally, the distributed DBMS approach is amenable to incremental upward scaling of database capacity. The distributed architecture permits a very large database to be supported on a collection of moderate size DBMSs, instead of on a single very large centralized site. And if more power should be needed to accomodate increases in database size or usage, this power can be added incrementally through the addition of new sites to the network. By contrast centralized systems

* This research was supported by the Advanced Research Projects Agency of the Department of Defense under contract no. N00039-77-0074. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

are often difficult to upgrade without major service disruption and conversion costs.

The above advantages, coupled with the maturation of the necessary technology, have now made the development of distributed DBMS a very active field. A number of distributed database implementations have been reported in the literature in the past year: Champine[12] describes six recently implemented distributed database systems; Foster[16, 17], Heller and Osterer[22], Pliner et al[37], and Wiseman[59] describe others.

These efforts represent the state-of-the-art in distributed DBMS implementations. Many of them are quite large and as a whole these implemented systems serve to validate the soundness of the distributed DBMS approach. Nevertheless, each of these currently implemented systems is a special purpose, one-of-a-kind system, designed to handle the particular data management needs of a single enterprise. Complementing these implementation efforts in distributed data management is a growing body of research and development activities. The major focus of this R&D is the development of general purpose, rather than special purpose, distributed DBMS. Such systems, like conventional non-distributed DBMSs, are seen as being available off-the-shelf, and as being able to solve a wide range of data management problems. No such general purpose distributed DBMS yet exists, and many technical problems remain to be overcome before such systems can come into existence. It is the intent of this paper to survey the significant R&D efforts aimed at overcoming these problems thus making possible general purpose distributed DBMS.

This paper reviews work in those areas that have been studied most intensively to date. It is by no means the case that the topics surveyed here cover all the problems that must be solved in developing practical and general purpose distributed DBMS; the field is a young and burgeoning one, and many problems have not yet been adequately addressed by any workers. The topics that are addressed here are the following:

*Section 2 — "Background" — presents concepts, assumptions, and issues that underly much of the work in the field;

*Section 3 covers the problem of synchronizing update transactions in a distributed DBMS; the section focuses in particular on the synchronization of transactions that update data that is stored redundantly in the distributed system;

*Section 4 discusses algorithms for efficient execution of transactions that access dispersed data; the key objective here is to allow transactions to reference data stored at multiple locations without incurring excessive communication costs and delays;

*Section 5 addresses "resiliency", i.e. the ability of the distributed system to continue operations despite component failures; the central difficulty in achieving resiliency is the need to ensure that database consistency is not violated as a result of component failures;

*Section 6 describes approaches taken to directory management, i.e. the management of catalogs and other data used by the system itself in (a) determining where data is stored in the network, and (b) how to parse and execute transactions against that data; and lastly

*Section 7 reviews work in the area of optimal file allocation; the goal of this problem is to allocate data among the nodes of the distributed DBMS so as to optimize factors such as response time, communication costs, storage costs, computation loads, and reliability.

The above areas cover most of the published r&d relevant to the development of general purpose distributed DBMS. One important topic which is not addressed is heterogeneous distributed DBMS, i.e. systems in which the nodes of the distributed DBMS may differ in data model and data access language. All of the issues presented in this paper apply to a heterogeneous systems as well, and in addition there is a substantial translation problem to be addressed. This type of problem has been tackled in a sub-field of database technology known as data translation and a substantial literature has grown up in this area. Selected readings on this topic are included in the reference list. The translation problem for heterogeneous systems represents an important application area for this technology but a review of the area is beyond the scope of this paper.

BACKGROUND

The field of distributed data management, being so young, is not yet based on a foundation of well developed and generally accepted concepts and assumptions. Different researchers in the field make different assumptions on issues like

- how data in the distributed DBMS is structured; and whether the distributed structure of the database is visible or invisible to users;
- the characteristics of the communication network that interconnects the distributed DBMS sites;
- the nature of "typical" distributed DBMS applications;
- the data model supported by the system; etc.

Often these assumptions are not verbalized in the papers, and often the different approaches taken by different researchers hinge on the disparity of their assumptions.

As background for the r&d survey that follows, this section reviews the major assumptions that underly work in the field. To the extent possible this section also identifies the impact of different assumptions on the work of different persons.

1. One area in which there is general agreement among the R&D community is general system architecture of a distributed DBMS. A typical architecture was illustrated in Figure 1 above. The system consists of a collection of independent computers each of which supports a portion of the overall database. These computers are interconnected via some type of communication channel, which is used to transfer data and control information.

Users are permitted to enter transactions at any of the computers; the users' transactions may reference data stored at a remote site in which case the distributed DBMS accesses the data by

means of the communication channel. In some cases, the problem of finding remote data is automatically handled by the system; in others the user must explicitly tell the distributed DBMS the site at which each datum is stored.

Many distributed DBMSs permit redundant data to be stored for reasons described below. In almost all such cases, the system is responsible for automatically updating all copies of the redundant data in response to users' update transactions.

These last two points -- the visibility or invisibility of data dispersal, and data redundancy -- are key questions and are discussed further below.

2. Within the general system architecture described above many different approaches can be taken regarding the distribution of data. For example

1. Each database site could contain a complete copy of the entire database; or
2. Each database site could contain a unique subset of the database which doesn't overlap the portions stored elsewhere; this partitioning of the database could be accomplished by assigning each file to a single site, or the unit of partitioning could be smaller;
3. A third choice is to store subsets of the database at each site as for choice #2, but to permit the various subsets to overlap; the impact of this generalization is that it allows redundant data to be stored in the database, meaning that the same logical data item is physically stored at several sites.

The first approach is proposed by Alsberg and Day,[3] Bernstein et al,[6] Ellis,[15] Thomas,[56,57] and Rothnie et al.[42] This so-called "fully redundant" approach, however, in clearly impractical for databases of useful size; realistically this approach should be viewed as a pedagogic simplification of the general "partially redundant" approach, #3 above. (The fully redundant approach is explicitly presented in this light in Bernstein et al[6] and Rothnie et al[43]).

The second approach does not permit redundant data. It turns out that the avoidance of redundant data simplifies the distributed DBMS problem significantly, and this approach has been taken by most special purpose distributed DBMS implementations to date. Unfortunately, though, much of the promise of distributed data management can only be achieved by permitting redundant data to be stored, and virtually all r&d efforts underway now do permit it.

This brings us to the third data distribution approach, the general partially redundant case, which is followed by most current r&d. The key advantages of this method over the simpler non-redundant one are the following:

1. Without redundant data the reliability goals of distributed data management can only partly be met. Without redundant data, the failure of a particular database site must cause the failure of all applications that require data stored there; even though many other nodes of the distributed DBMS might be up and running, these nodes are of no value to applications requiring data from the failed component. I.e. from the point of view of these applications, the distributed DBMS will have suffered total failure from the failure of a single component.
2. Without redundant data the choice as to where a given subset of the database is stored is an all-or-nothing choice. If, for example, some data were accessed equally from Los Angeles and from New York, there might be no good choice for its location in a non-redundant system.
3. Finally, the redundant data approach permits more flexibility in increasing database capacity to support very large databases. Portions of the database which are accessed frequently can be stored at many small sites using relatively fast secondary storage. Other portions of the database that are needed only occasionally could be stored at an archival site on inexpensive, but slow, tertiary storage. Moreover the redundant approach allows additional database sites to be added to accomodate increases in database activity, whereas in a non-redundant system increases in activity against a selected subset of a large database could require an upgrade of the site at which that subset were stored.

For these reasons, most research efforts that are ongoing now assume that redundant data is permitted. This includes work by Alsberg et al,[1,2] Bernstein et al,[7] Rothnie and Goodman,[42] and Stonebraker and Neuhold.[53]

3. Another aspect of data distribution, other than the question of data redundancy, concerns the invisibility of data distribution to users. In much of the literature, it is assumed that users are not required, nor able for that matter, to specify

a. the location of data for access operations, or
b. the location of redundant copies of data for updates,

In these systems users enter queries and updates into the distributed DBMS in precisely the same manner as they would in a non-distributed system. The motivation for keeping data distribution invisible is that distribution is a database design issue similar to the choice of secondary indices, and for reasons of data independence users should not and need not be aware of it. With respect to redundant updates the motivation for invisibility is even stronger: consistency of the database depends on keeping all redundant copies of data up-to-date and identical to each other (see Section 3); if responsibility for this function were entrusted to users serious integrity problems could ensue.

Of the major current r&d efforts the only one where data distribution is not strictly invisible to users is Stonebraker and Neuhold.[53] The system described by these authors supports two modes, one in which distribution is invisible and one in which users must specify data locations in their transactions. The motivation for the latter mode is to improve performance.

4. Assumptions regarding characteristics of the communication medium used by the distributed DBMS vary widely among researchers in the field, and those assumptions are very rarely stated in the published work. The types of communication channels that are most frequently postulated are

1. "Ethernet"[35]-like networks -- high band width, low delay networks suitable over short ranges;
2. Packet switch networks, of which the Arpanet[10,34] is a frequent representative; packet switched networks have both lower bandwidth and higher delay than Ethernet-like media, but of course are usable over much longer ranges;
3. A final type of communication channel that is common in the distributed DBMS literature is point-to-point leased circuit configurations; such networks would typically support lower bandwidth than the Arpanet, but would also incur lower delay.

An additional network concept that one should expect to see in the forthcoming literature is broadcast networks, either radio broadcast networks such as Aloha[8,16], or broadcast networks employing satellites.

These types of communication channel vary in bandwidth over a range spanning 3 orders of magnitude and in delay over 6 orders of magnitude! Needless to say the assumptions that are made here have a significant effect on the problems one must face in developing a distributed DBMS. Among the issues affected by this assumption are:

- the feasibility of propogating control information to all database sites in real-time;
- the feasibility of posting updates to all copies of redundant data in real-time;
- the relative cost of moving data from site-to-site in the processing of complex queries, vs. the cost of performing extra "local" computations.

Few of the published works explicitly state the type of media that is assumed to be in use. Most of the literature takes as a given one or another of the types of network mentioned here, and proceeds to design a distributed DBMS within the confines of that network environment. The converse approach of designing a network that is ideally suited for a distributed DBMS has not yet been addressed, although clearly such work could be fruitful.

5. While all the R&D efforts surveyed in this paper are concerned with general purpose distributed DBMS, they are nonetheless affected by their perception of what a "typical" application is likely to be. Virtually all approaches assume that typical applications will involve real-time, interactive transaction processing. However, approaches differ with respect to issues such as

- will typical transactions manipulate large amounts of data or small amounts?
- how large is the database likely to be?
- how "valuable" is each transaction? If some system component fails, is it crucial that the absolute minimum number of transactions possible be aborted, or can the system be somewhat more cavalier?
- to what extent will access to the database be geographically localized?

One trend in the distributed DBMS literature seems to be motivated by distributed operating system, rather than distributed database, objectives. For those systems typical databases are small, consisting of state information on process, etc. in the operating system. Typical transactions tend to be quite simple and short; generally, each reads and/or modifies a few words of state information for some process. Moreover it is crucial in these systems that operations continue with absolute minimum disruption upon failure of individual components.

The characteristics of true distributed database systems are quite different from the above. Databases tend to be large in these systems, possibly ranging into the trillion bit size. While update transactions tend to be relatively simple, database retrievals are often quite complex and involve large amounts of data.

Although it is clear that one's perception of "typical" applications has a great effect on one's choice of distributed DBMS designs, few of the published papers explicitly state their presumed application context. Consequently a caveat emptor situation is in effect to some extent, and the reader is cautioned to beware in evaluating competitive distributed DBMS designs.

6. A final area of background information to be presented here concerns the data model supported by the system and the language provided for accessing the data. These issues do not so much affect the problems that must be addressed in distributed DBMS research; they tend instead to affect the range of solutions that are feasible for solving those problems.

One simple example will be presented to illustrate the point: in a relational DBMS, queries are expressed in a non-procedural mathematical form which can be transformed by well-known techniques into other equivalent queries. This provides a direct avenue for transforming relational queries as entered by the user into equivalent forms that can be executed more efficiently with a given distribution of data among the nodes of the distributed system.

By contrast, in a CODASYL or hierarchical DBMS with a host data manipulation language, a complex query would be expressed as a complex host language program; and, in general, the problem of transforming programs into equivalent programs is quite difficult.

This concludes the presentation of background concepts and material in this paper. The remainder of the paper proceeds to survey r&d efforts addressing the major technical problems of general purpose distributed DBMS development. We start by considering first the problem of synchronizing updates to redundantly stored data items.

SYNCHRONIZING UPDATE TRANSACTIONS

In developing a distributed DBMS perhaps the

single most difficult problem is the synchronization of update transactions. It is necessary to develop techniques for controlling concurrent transactions so that

a. database consistency is preserved, while
b. excessive overhead in propogating control information among the nodes of the distributed DBMS is avoided.

Moreover, for the reasons enunciated in the preceding section, most general purpose distributed DBMSs allow data items to be stored redundantly at multiple database sites; whatever update synchronization methodology is used by the distributed DBMS must also ensure that all redundant copies of data get updated correctly and consistantly.

The problem of distributed DBMS update synchronization has been addressed by numerous authors including Alsberg et al,[1,2] Alsberg and Day,[3] Bernstein et al,[6,7] Ellis,[15] Johnson and Thomas,[25] Lampson and Sturgis,[28] Rosenkrantz et al,[39] Rothnie et al,[43] Rothnie and Goodman,[41,42] Stearns et al,[52] Stonebraker and Neuhold,[53] and Thomas.[56,57]

The simplest method for synchronizing distributed updates is to lock those portions of the database being read or written by active transaction. Indeed locking is the usual mechanism employed for update synchronization in conventional, non-distributed DBMSs. However in a distributed database environment an appreciable and often intolerable delay is introduced as locking information is propogated to the many computers in the database network.

For example, a straightforward distributed locking algorithm described by Rothnie and Goodman[41] requires 5n intercomputer messages in order to synchronize a transaction among n nodes of a distributed database system:

n lock request messages, n lock grant messages, n messages to transmit the update, n update acknowledgements, and n messages to release the locks.

Furthermore, the delay in executing the update is lengthy, engendering:

the maximum delay encountered in setting the locks

+

the maximum delay encountered in performing the update

The communication delay that would be encountered if the system used a communication network such as the Arpanet for these messages could easily be as much as .1 - 1 second. This delay is some 2 to 3 orders of magnitude greater than the delay typically encountered when setting locks in a centralized system.

For these reasons the straightforward locking approach is inadequate for a general purpose distributed DBMS and other methods must be sought.

A variety of solutions to this problem have been proposed. Certain of these solutions are closely related to the locking approach in concept, seeking to find more efficient algorithms for propogating lock information through the database network. For example, Lampson and Sturgis[28] describe a method involving "two phase commit" in which only 4n messages are needed to propogate locking information to n sites of a distributed DBMS. It is possible to reduce this cost even further by "piggybacking" "update transmission" messages on "lock request" messages, thereby reducing the cost of locking to 3n. (Although this only reduces communication volume if the update transmission messages are short or if most lock requests are granted) Ellis[15] describes an algorithm which further reduces communication volume to 2n. This method is similar to Lampson and Sturgis's two phase commit approach, but uses a sequential daisy chain communication procedure for propogating synchronization messages from site to site. While this method reduces communication volume by 50% over the Lampson and Sturgis method, its sequential communication procedure greatly increases communication delay.

As a final example, Thomas[57] has described a solution that employs daisy chain communication and a "voting" protocol to enable lock setting by a majority of database sites rather than requiring unanimous approval. This method reduces communication volume to about 1.5n, but like Ellis's algorithm it introduces lengthy communication delays.

While the savings achieved by the above algorithms are substantial, in many cases these methods still require a large amount of inter-computer communication in order to perform updates. It appears, therefore, that these methods will perform unacceptably in networks containing large numbers of sites and high transaction volumes.

Other approaches seek to reduce synchronization costs by introducing some degree of centralized control in the distributed system. Alsberg et al,[1,2] Alsberg and Day,[3] and Stonebraker and Neuhold[53] have proposed "primary site" methods which require that all update activity for a given file (or in some cases, sub-file) be funneled through a single database site called the primary site. This approach appears to be quite good for simple database applications in which it is possible to effectively partition database activity by geographic region. In more complex situations, though, this method cannot avoid the need for global database locking with concomitant high communication cost and delay. In particular, transactions which access multiple files with different primary sites must perform locking in order to guarantee that the files are consistent with each other. This is true for retrieval transactions as well as for updates!

Another approach to the update synchronization problem which is qualitatively different from the preceding methods is described by Bernstein et al,[6,7] Rothnie et al,[43] and Rothnie and Goodman.[42] Rather than merely trying to improve the efficiency of global database locking, this method seeks to avoid global locking whenever possible. The method is based on a formal analysis of the ways in which transactions in a distributed DBMS can interfere with each other, and the ways in which this interference can be avoided. The methodology described by these authors is being implemented for SDD-1, A System for Distributed Databases, under development by Computer Corporation of America.

The SDD-1 methodology achieves update synchronization by means of several different "synchronization protocols" which vary in cost and which offer varying levels of synchronization control. Each synchronization protocol is an algorithm that specifies what to do to ensure the correct processing of a given transaction. The most efficient of the protocols specifies no inter-computer synchronization whatever; a transaction run under this protocol need only perform local locking to ensure the atomicness of its read and write operations at each individual site. Each of the other protocols introduces some degree of non-local synchronization, with the strongest protocol achieving even greater control than global database locking.

Not all transactions may be run under the most efficient protocol, of course. Correct operation of the system depends critically on the correct selection by the database system of the protocol to use in processing each transaction entered by users.

The decision as to which transactions must use which protocols is made off-line, for example during database design. The decision process begins with the database administrator defining classes of transactions that are commonly executed in the database application. Then using rules presented in Bernstein et al[6,7] and Rothnie et al,[43] the defined transaction classes are algorithmically analyzed, yielding a specification of which transaction classes must use each protocol. These results can then be summarized in tables which are stored at each database site for later use at run-time.

The run-time protocol selection function operates by first mapping each transaction entered into the DBMS into the class(es) of which it is a member. Then for each class the function looks up the correct protocol for transactions in that class, using the table stored at the local database site. If the transaction is a member of several classes, the selection function chooses the most efficient one; if the transaction is not a member of any defined class, the function selects the strongest protocol defined for the system.

It is important to note that the run-time operation of the protocol selection function does not itself require any inter-computer communication. This is because all the "intelligence" needed to ensure correct synchronization of transactions in each class is already "compiled into" the protocol selection tables.

It appears that the SDD-1 update methodology permits faster and lower cost execution of update transactions than can be achieved using any alternative method yet proposed.

DISTRIBUTED QUERY PROCESSING

In this section we consider the problem of processing a query which accesses data at multiple sites of the distributed DBMS. How does this problem differ from query processing in a non-distributed DBMS? There are two significant differences to consider: First, there is a substantial new element of processing delay, the time required to communicate among the sites involved in the query. Second, there is an opportunity for parallel processing since there are several computers involved in handling the query. The following example illustrates these differences.

Consider the relations of figure 2. These relations provide information about supply relationships among certain suppliers and the projects they supply. Projects contains project identifiers (J#) and the location (J-Location) of each project. Suppliers indicates the identifiers of suppliers (S#) and their locations (S-Location). Parts tells the name (P-Name) and length (P-Length) of each part. Finally Supply ties these objects together by indicating which suppliers (S#) supply which parts (P#) to which projects (J#). Figure 2 also indicates the size of each relation and the site where each is stored.

Relation Name	Attributes	Bits per Tuple	Number of Tuples	Site
Projects	J# J-Location	100	10,000	A
Suppliers	S# S-Location	100	1,000	B
Parts	P# P-Name P-Length	100	100,000	C
Supply	S# P# J#	100 100	1,000,000	A

Figure 2 Example Relations

We will assume an Arpanet-like communications facility with a transmission delay of 1 sec and a bandwidth of 10,000 bps. As we shall see, the relatively long transmission delay has a substantial impact on the choice of a strategy for query processing: it causes the transfer of data between processes on different machines to be much more efficient if it is accomplished as a continuous stream rather than as a series of separate interactions.

Now, consider a query which selects the identifiers of projects in Boston which use 10 inch bolts. A DSL-ALPHA expression for this query is:

```
Range  J  Projects
Range  Y  Supply Some
Range  P  Parts Some
Get W J.J# where J.J-Location="Boston" and
                J.J#=Y.J# and Y.P#=P.P# and
                P.P-Name="Bolt" and P.P-
                Length=10
```

In order to calculate the costs of various strategies for processing this query we need to estimate the sizes of three intermediate results:

1. The number of projects located in Boston -
   Count (J where J.J-Location="Boston")
   Estimate=1,000
2. The number of supply tuples for projects located in Boston -
   Count (Y where Y.J#=J.J# and J.J-Location="Boston") Estimate=100,000

3. The number of parts which are 10 inch bolts -
Count (P where P.P-name="Bolt" and P.P-Length =10) Estimate=10

With this preliminary information we will proceed to consider (briefly) six alternative strategies for handling this query, and we will calculate the communications delay incurred by each strategy. Delay in seconds will be computed by the formula:

Delay = Number of Interactions x 1 sec
+
Volume/10,000 bps

Strategies 1 and 2 involve an initial step of rendering the query local by moving all relations involved to a single site. When this step is completed the query is processed by some non-distributed algorithm.

Strategy 1 involves moving the Parts relation to site A. This entails 1 interaction and 10^7 bits of total volume for a delay of 10^3 seconds or 16.6 minutes. Strategy 2 involves moving Supply and Projects to site C at a cost of 2 messages and about 10^8 bits of data transfer for a delay of more than 2.7 hours.

Strategies 3 and 4 mimic a strategy which is sometimes used in single site relational query processing called tuple substitution. In this scheme a multi-variable query is processed as a sequence of 1-variable queries by repeatedly substituting tuples for certain variables into the query.

Strategy 3 will substitute tuples from Supply and Project to generate a sequence of queries to process against the Parts relation. For example, if the Supply tuple (7036, 152475, 1567) and the Project tuple (1567, Boston) were substituted into the original query the resulting 1-variable query to process at site C would be:

(P where P.P#=152475 and P.P-name="Bolt" and P.P-Length=10)

In essence the process has found a project in Boston which uses part 152475 and we are now asking if this part is a bolt. Using the estimated sizes provided above there will be 100,000 questions of this form to ask. (This number can be reduced by using query feedback techniques of the form described by Rothnie[40] but these will not be considered here.) Each of these queries to site C will require a separate interaction in each direction. This will generate a delay of 55.5 hours.

Strategy 4 involves substitution of Parts tuples into the original query to produce 2 variable queries involving the relations Supply and Projects. Such queries can be processed entirely at site A. Our estimates indicate that 10 such queries will be generated for a delay of about 20 seconds.

Strategies 5 and 6 involve the initial computation of subsets of the relations at sites A and C and then moving the subset at one of these sites to the other one to complete the query processing.

Strategy 5 computes at site A all of the P# and J# pairs for projects located in Boston. This relation is then moved to site C to determine which of the P#'s correspond to 10 inch bolts. At site A this involves processing the query:

Get W (J.J#, Y.P#) where
(J.J#=Y.J# and J.J-Location="Boston")

W will contain about 100,000 tuples of (say) 100 bits each. This will be transmitted to C in 1 interaction with a delay of about 10^7 sec or 16.6 minutes.

Strategy 6 involves the computation at site C of the P#'s for parts which are 10 inch bolts and transmitting these P#'s to site A. Our estimates indicate that there will be 10 such values and hence, that they will be transmitted to A in less than a second.

Figure 3 summarizes the delay incurred by each of these strategies.

Strategy	Description	Messages	Volume	Delay(sec)
1	Move Parts to A	1	10^7	10^3
2	Move Supply and Projects to C	1	10^8	10^4
3	Substitute tuples from Supply and Projects	2×10^5	10^7	2×10^5
4	Substitute tuples from Parts	20	10^3	20
5	Move restricted join of Supply and Projects to C	1	10^7	10^3
6	Move restriction of Parts to A	1	10^3	10^{-1}

Figure 3 Communications Delay for Various Example Strategies

There are four points to observe from this example:

1. There is a very great variation in communication cost among a set of plausible distributed query processing schemes.
2. The absolute magnitudes of the communication delays are very great for the poor strategies and seem likely to dominate the total query processing delay in these cases.
3. The two dimensions of communication delay, end-to-end delay and bandwidth, are both important in choosing a distributed query processing strategy. For the parameters used in this example it is better to employ a strategy which transmits a batch of data at once

(as in strategies 5 and 6) than small amounts of data in each of many separate messages (as in strategies 3 and 4).

4. While the communication delays we have computed do not reflect this, the best strategy (Strategy 6) provides opportunities for parallel processing which can reduce the total elapsed time for handling this query. Specifically, at site A we can compute the (JF, PF) pairs for Boston projects while at site C computing the set of PF's for 10 inch bolts. Then when the PF's are moved to site A the solution to the query can be obtained by taking the join of these intermediate results. For some queries this effect will be significant and can result in elapsed processing times for distributed queries which are actually less than for single site queries.

The literature contains few reports of work in this area of distributed query processing. Results which have been published describe algorithms which are similar in their approach to algorithms invented for use in handling relational queries. This approach involves:
first, the definition of a family of strategies which can be applied to compute the result of a query and,
second, an optimization scheme to choose the least costly member of this family. Each of the schemes to be mentioned here adopts this approach.

Wong[60] has proposed an algorithm for use in SDD-1 in which each member of the family of strategies consists of a sequence of two types of actions: local processing performed in parallel at several sites and parallel moves of subrelations from site to site. This sort of strategy is rather similar in concept to the decomposition scheme which Wong developed for use in INGRES.[54,61] Wong's decomposition attempts to handle a difficult problem, (multi-variable relational queries) as a sequence of easier problems (one-variable queries) and actions which transform the multi-variable query into one-variable queries (tuple substitution). For SDD-1, the Wong approach is to solve a difficult problem (distributed query processing) as a sequence of easier problems (local query processing) and actions which transform the distributed query into local queries (subrelation moves).

The optimization scheme in SDD-1 begins with a proposed solution consisting of performing all local processing which can be performed without any data moves and then moving the results to a single site where the query processing is completed. The single site is selected to minimize the costs of the data moves.

In our example strategy 6 would be the starting point solution for this algorithm.

From there the optimization scheme looks for an improvement involving one set of moves and local processing to be performed prior to the movement of subrelations to the site selected in the initial solution. Sometimes this prior move will reduce the subrelations to be moved to the final site substantially and result in a lower cost for the complete query.

If some improvement is gained, the optimization scheme is applied recursively to determine if any further gains can be obtained. The algorithm can be characterized as a "greedy" one because it always looks for immediate gains and and hence will always terminate on a local optimum but not necessarily a global one.

Of the distributed query processing algorithms reported in the literature, Wong's is the only one to consider communications costs as a key element of query processing cost. Hence it is the most strongly oriented toward the distributed DBMS environment. As we shall see the other algorithms represent smaller perturbations from local query processing techniques.

Schneider[45] has attacked a broader query processing problem than the other authors we will consider here. He treats the issue of processing a query in a network where the database may be distributed over a set of dissimilar data handlers. The proposed solution uses the data representation tools of DIAM to deal with heterogeneity and its generalized search path selection schemes to handle the optimization problem. The technique does not consider communication cost issues. Schneider also deals with the query transformations which are necessary to process a query at a local site when the data there represents a projection, restriction or join of the relation which the user has referenced.

Stonebraker and Neuhold[63] have proposed an extension to INGRES to deal with distributed databases. This extension uses a straightforward modification of Wong's decomposition to deal with distributed query processing. The scheme preserves tuple substitution as a basic strategy and hence is vulnerable to the type of phenomenon observed in strategies 3 and 4 of the example. While the volume of communication may not be particularly large the need for a long series of separate interactions across the network can lead to long processing times due to transmission delay. It is this effect which Wong attempted to avoid in replacing tuple substitution by subrelation moves in SDD-1.

HANDLING COMPONENT FAILURES

One of the key motivations for distributed database systems is a requirement for high database availability, that is, a need to ensure that a database is nearly always accessible. Distributed database systems seem to offer this characteristic since availability is not limited by the reliability of any single component but rather by the reliability of combinations of components (processing nodes and communication links in the network). These combinations can be configured to achieve arbitrarily high availability. Belford et al[5] presents an analysis which determines availability as a function of the configuration of a distributed database system.

In order to achieve this reliability potential, however, it is necessary that the distributed system be able to cope with the failures of individual components and continue operation. In this section we outline the problems inherent in constructing a system of this sort and briefly

describe the solutions which have been proposed. The central problem in reliable operation of a distributed database system is maintaining database consistency in the presence of failures during update transactions. Such failures threaten to destroy the "atomicness" of these transactions by causing the update to be only partially accomplished in the database. Consider, for example, a transaction T_1 which updates portions of a database stored at nodes N_1 and N_2. If a failure occurs during the execution of T_1 which prevents the update from being recorded at N_2 then the database has been made inconsistent. A later transaction which reads the results of T_1 will see an anomolous database with unpredictable results. The system updating algorithms must therefore be aware of the possibility of component failure and avoid these partial results.

The avoidance of database inconsistencies of this type is the major objective of failure handling mechanisms but it is clear that there are other desiderata which are important to the design of these algorithms. Perhaps first on the list is efficiency. We desire that the responsiveness and throughput of the system be "good" both during normal operation and in the presence of failure. This is an important consideration because it eliminates a very simple mechanism which preserves consistency but introduces intolerable delay in the presence of failures. This simple algorithm specifies that any transaction which would normally communicate with a failed node must wait until that node has recovered from the failure before it can proceed. Clearly this sort of approach can lead to indefinite delays, and is therefore unacceptable. An effective distributed DBMS must employ a different type of mechanism: one which attempts to execute to completion any transaction which can proceed without a resource (such as a data item) which is exclusively available at the failed node.

The problem to be solved by efficient failure recovery algorithms can be partitioned into the following subproblems:

1. Reliable broadcast - Consider a distributed database system which is executing a transaction T. Suppose that the execution is being "controlled" from one node in the network, N_1, and that the results of T will have an effect on the fragments of the database stored on nodes N_2, N_3 and N_4. Let us further suppose that nodes N_2, N_3 and N_4 have received instructions indicating what changes are to be made in their database fragments but are awaiting a signal, W, from N_1 before actually writing the changes. In order to insure that the results of T are completely installed in the database the system must guarantee that W will either reach everyone of N_2, N_3 and N_4 or none of them. This guaranteed delivery at multiple sites we will refer to as reliable broadcast.

Requirements for this sort of characteristic arise frequently in the distributed database systems designed to date. It is a difficult characteristic to achieve because the communication networks supporting these systems typically offer pair-wise communication only. Thus, in order to send W to N_2, N_3 and N_4, N_1 must send three distinct messages. There is, of course, a finite probability that N_1 will fail after the message has been sent to N_2 but not before it has been sent to N_4. If this occurs then reliable broadcast has not been achieved. This problem can be handled in a variety of ways. Lampson and Sturgis[28] try to minimize the reliable broadcast window (e.g. the time between sending W to N_2 and sending W to N_4) and hence minimize the likelihood of a failure during that inconvenient period. If a failure does occur during that time the database is not left inconsistent but locks set during the execution of T will remain set until the failed node is recovered.

In the SDD-1 design, Hammer and Shipman[21] try to insure that reliable broadcast is achieved even when N_1 fails during the broadcast window. This is accomplished by requiring that the first few recipients of W also send the message on to the other addressees. (This requires that all recipients be capable of handling duplicate messages.) In addition, each node is warned when a reliable broadcast is about to be sent and any node which does not receive the message in a reasonable period will ask the other addressees if they have received it. By selecting the parameters of this scheme appropriately the likelihood of a breakdown in the reliable broadcast can be made arbitrarily small.

2. Operation with missing nodes - A second subproblem in failure handling is continuing operation of the system when one or more nodes in the network are known to be down or otherwise unavailable. The objective here is to avoid having the absence of these nodes cause transactions to be delayed until the missing nodes are recovered. All of the existing system designs try to accomplish this objective by acting (almost) as if the missing nodes never existed. Non-existent nodes of course cannot be allowed to cause any delays. It is necessary however to take some note of these failed nodes in order to prepare for their recovery and to deal with any incomplete actions they have left behind.

Thomas[37] Ellis[15] and Hammer and Shipman[21] each prepare for a failed module's recovery by continuing to send relevant updates to the node so that they can be applied when the node is restarted. As we indicate below, these updates are held at some other site pending the recovery.

Another way in which the Thomas algorithm recognizes the existence of a failed site is in determining the number of nodes in the majority which is required to finally accept a transaction. This majority is computed on the basis of the full original network and not only on the currently active sites.

The Lampson and Sturgis scheme will sometimes have a lingering recollection of a failed node in the form of locks which cannot be released until the node recovers.

3. Restarting a node - A third subproblem is re-integrating a node into the system when the cause of the node's failure has been corrected. Since the fragment of the database stored at the node may be out of date, it is necessary for the node to find out what's happened during its absence prior to continuing its active participation in the system. The principal mechanism for accomplishing this task might be called "persistent communication" after the persistent processes

of Sutherland. [53] Persistent communication guarantees the delivery of a message to a node even if the addressee is down when the message is sent and even if the sender is down when the addressee recovers. This sort of mechanism is a very clean way of achieving recovery since the restarting node simply acts on its old messages in almost the same way it does in normal operation. Thus, few new facilities are needed to accomplish node recovery. Thomas, Ellis, and Hammer and Shipman use forms of persistent communication for restarting failed nodes.

4. Failure detection - The mechanisms discussed under "Operation with missing nodes" and "Restarting a node" both depend on knowing when a node has failed. The methods used for detecting failure are all variations on a single time-honored technique called time-out. Under this scheme, a node which is suspected of having failed is probed with a message to which an active node will respond. If the response does not arrive within some prescribed time period the probed site is assumed to have failed.

5. Partitioning - The final subproblem is a serious one for which no adequate technical solution has been proposed. This problem arises when communication failures break all connections between two or more active segments of the network. When this occurs each isolated segment will continue its operation, including processing updates, but there is no way for the separate pieces to coordinate their activities. Hence the fragments of the database in the separated pieces will become inconsistent. This divergence is unavoidable if the segments are permitted to continue general updating operations and in many situations it is essential that these updates proceed. However, a problem arises when the partition is repaired and communications resume. How are the inconsistencies to be reconciled?

This problem is not amenable to solutions which attempt to construct a new and consistent database by somehow interleaving the transactions run in the seperate segments. This approach will not work because some of the transactions run in the partitioned network are simply not correct in the context of a connected network. Consider the following example.

Suppose there is a distributed DBMS serving a fleet of naval ships with nodes located on each ship. The database includes the location of every ship in the fleet. Further, suppose that a portion of the fleet becomes separated from the rest and communications are lost. The separated sub-fleet might decide to assume that the rest of the fleet will move according to the original operations plan and update its database accordingly. Meanwhile the main body might choose a different course and update its database with the correct locations. Furthermore, some guess about the locations of the separated ships might be made and recorded. When the fleet is re-integrated the database will be inconsistent. The appropriate action to take at this point is to construct a new consistent database by taking certain fragments from each of the separated segments and discarding the rest.

In general, the correct action to take when a partition is repaired depends on the semantics of the database, the topology of the partition, and the transactions which have been run during the partition. The integration process is likely to require human decision making assisted by automatic conflict detection and some limited automatic conflict resolution.

DIRECTORY MANAGEMENT

An important characteristic of modern database management systems is that they employ a directory (often called a schema) to define the database being managed. This frees the user or application program from the need to supply this information, and it simplifies many types of database structural changes. A directory typically contains 4 types of information:

- logical structure definition - e.g. the names of relations and their domains;
- physical structure definition - e.g. data field formats, inverted fields;
- file statistics - e.g. size; and
- accounting data - e.g. who has accessed the file, who owns the file.

For a distributed database we must add an additional category of directory information: the location of each piece of the database in the network.

The DBMS must have access to this information in order to parse user requests, choose and execute an accessing strategy, and account for the resources used. The following question therefore arises: where should the directory be stored?

There are many possible answers to this question and the best answer varies with the database and particular accessing pattern. Furthermore within a single system the entire directory need not be handled uniformly. Some types of information, like the logical structure definition and location information will be needed frequently and may be handled differently from less frequently accessed data, like accounting data.

The principal directory management schemes can be divided into two categories: non-redundant approaches in which each directory entry is stored in only one location and redundant approaches in which each entry may appear in several places. Among the non-redundant schemes the principal alternatives are the following:

- centralization - The complete directory is stored at exactly one site. This requires access to the directory site for every retrieval or update to the directory.
- distribution - Each site has the directory entries for the data stored at that site. Completely local access can proceed using the local directory. Any request requiring access to remote data must be broadcast so that other sites can determine whether or not they have relevant data.
- combinations - An intermediate approach is to partition the network into several pieces and employ a centralized directory in each piece.

The redundant approaches include the following alternatives:

centralization - A redundant but centralized approach might be a combination of the centralized and distributed non-redundant alternatives. That is, each site has the entries for its local data but a central site has a complete directory. In this way non-local references can appeal to a single site to determine the location of the remote data.

distribution - Each site has the complete directory stored locally. All user transactions can be started into execution without a remote reference, but all directory updates must be posted to every site.

combinations - Many combinations are possible. The most general would permit an arbitrary subset of the directory to exist at each site. This permits a great deal of flexibility but it requires a more complicated "directory directory" to tell the system what piece of the directory is stored where.

The factors which determine the choice of a directory management scheme are these:

directory retrieval frequency - High frequency of directory retrievals encourages redundant directories and distribution so that most retrievals will not require access to remote nodes of the network. Since every user transaction requires a directory retrieval, directory redundancy is nearly always desirable.

directory update frequency - High frequency discourages redundancy and encourages centralization. This is because every copy of the directory must have the change posted. Since this requires the kind of concurrency control discussed in Section 3, directory updates can be expensive to process.

reliability - A requirement for high reliability encourages redundancy and distribution for obvious reasons.

Stonebraker and Neuhold[53] propose to use a redundant directory approach for INGRES. In this scheme each site stores the directory entries for local data as well as locator information for remote data. In addition they suggest the use of a local cache at each site which holds the directory entries for recently accessed data.

Rothnie and Goodman[42] propose a more general mechanism for SDD-1. They observe that the storage options and criteria of choice for directories are exactly the same as those of user files. Hence they suggest that directories be treated exactly as the user data. This permits arbitrary subsets of the directory to be stored at each site. It also makes other system features such as security, integrity, reliability and concurrency control available for directory management. Decisions concerning allocation of the directory to physical sites can therefore be deferred until database design rather than being built into the structure of the distributed DBMS itself. The decisions at database design time are amenable to the techniques discussed in the next section: File Allocation.

DATABASE DESIGN

The discussion concerning directories in the preceding section addressed the issue of system access to a description of a database. In this section we consider the problem a user faces in creating such a description: the problem of database design. In particular, we will discuss the database design issue peculiar to the distributed environment, namely the allocation of pieces of the database to sites in the network. This problem is frequently referred to as file allocation.

There is a considerable literature concerning file allocation[11,13,29,30,32,58] much of which is reviewed by Levin and Morgan.[30] These research efforts have applied classical mathematical programming techniques to variants of the following problem:

Given: a description of user demand for service stated as the volume of retrievals and volume of updates from each node of the network to each file;

Given: a description of the resources available to supply this demand stated as the network topology, link capacities, and costs, and the node capacities and costs;

Determine: an assignment of files to nodes which does not violate any capacity constraints and which minimizes total costs.

The variations on this problem which have been explored include:

- consideration of time varying and uncertain demand;
- consideration of the dual problem of constraining costs and determining capacities; and
- the use of heuristics to reduce the computational complexity of finding an acceptable solution.

This work on file allocation has led to substantial understanding of the database design problem stated above. Unfortunately, however, this problem is only a small piece of the file allocation problem in a distributed DBMS of the sort discussed in this paper. There are three specific shortcomings of the existing body of research in this field which limit the usefulness of these results in the context of our problem:

1. The user demand model is stated as a set of requirements to access a given file from a given node. This model does not adequately reflect user demand for database access involving more than one file. Consider the example problem treated in the section on dispersed data access. This problem involved the join of relations at two sites. The bulk of the communication costs for that problem incurred between the nodes which stored the relations involved. This cost is not reflected in a model which considers only user node to file node communication. In particular the model cannot represent the

fact that a file allocation scheme which places all of the relations involved in the query at a single site will be cheaper for this query than one in which they are distributed. This is a serious shortcoming since it seems likely that the clustering of relations into collocated groups will be a more important determinant of accessing cost than the assignment of each group to a particular node.

2. A second serious problem with existing file allocation results is the complete neglect of synchronization costs in updating redundantly stored data. As the previous sections indicated, this cost is likely to dominate update costs (particularly for cost measured as delay), it will vary substantially with the synchronization scheme, it will vary with the mix of update traffic, and it will vary in a complex way with the database design.

3. The assumption that complete files should be the unit of assignment of data to nodes seems inflexible. There are many situations in which permitting a vertical or horizontal partitioning of files will reduce data accessing and storage costs. Furthermore two prototype distributed DBMSs currently being developed will permit this sort of partitioning. Of course, if a sub-file partitioning were given the existing results could be used to distribute them. However, this begs the question of determining what the partition should be. Answering this question seems to be an important element of database design.

The discussion above is not intended to say that existing results in file allocation are useless. On the contrary, when user behavior is accurately modeled as interaction with single files, these well developed results are valuable indeed. However, in the context of a modern DBMS in which multi-file transactions are important, existing research results do not provide adequate tools for database design.

REFERENCES

1. Alsberg, P.A.; Belford, G.G.; Bunch, S.R.; Day, J.D.; Grapa, E.; Healy, D.C.; McCauley, E.J.; and Willcox, D.A. Synchronization and Deadlock, CAC Document Number 185, CCTC-WAD 6503, Center for Advanced Computation, University of Illinois at Urbana-Champagne, Urbana Illinois, March 1, 1976.

2. Research in Network Data Management and Resource Sharing: Final Research Report, CAC Document Number 210, CCTC-WAD Document Number 6508, Center for Advanced Computation, University of Illinois at Urbana-Champagne, Urbana Illinois, September 30, 1976.

3. Alsberg, P.A.; and Day, J.D. "A Principle for Resilient Sharing of Distributed Resources", Report from the Center for Advanced Computation, University of Illinois at Urbana-Champagne, Urbana Illinois, 1976. (Also accepted for proceedings of the Second International Conference on Software Engineering.)

4. Bakkom, D.E.; and Behymer, T.A. "Implementation of a Prototype Generalized File Translator", Proceedings of 1975 ACM SIGMOD International Conference on the Management of Data, 1975, pp. 99-110.

5. Belford, G.G.; Schwartz, P.M.; and Sluizer, S. The Effect of Backup Strategy on Database Availability, CAC Document Number 181, CCTC-WAD Document Number 5515, Center for Advanced Computation, University of Illinois at Urbana Champagne, Urbana Illinois, February 1, 1976.

6. Bernstein, P.A.; Goodman, N.; Rothnie, J.B.; and Papadimitriou, C.A. "Analysis of Serializability in SDD-1: A System for Distributed Databases (The Fully Redundant Case)", Proceedings First International Conference on Computer Software and Applications (COMPSAC 77), IEEE Computer Society, Chicago Illinois, November 1977. (Also available from Computer Corporation of America, 575 Technology Square Cambridge, Massachusetts 02139 as Technical Report No. CCA-77-05).

7. Bernstein, P.A.; Rothnie, J.B.; Shipman, D.W.; and Goodman, N. The SDD-1 Redundant Update Algorithm (The General Case), Technical Report No. CCA-77-09, Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139, August 1, 1977.

8. Binder, R.; Abramson, N.; Kuo, F.F.; Okinaka, A.; and Wax, D. "ALOHA Packet Broadcasting - A Retrospect", Proceedings AFIPS 1975, National Computer Conference, AFIPS Press, Vol. 44, 1975, pp. 203-215.

9. Bires, E.W.; and Fry, J.P. "Generalized Software for Translating Data", Proceedings AFIPS 1976 National Computer Conference, AFIPS Press, Vol. 45, 1976, pp. 889-899.

10. Specification for the Interconnection of a Host and an IMP, Report No. 1822, Bolt Beranek and Newman Inc., Cambridge, Massachusetts January 1976 revision.

11. Casey, R.G. "Allocation of Copies of Files in an Information Network", Proceedings AFIPS 1972 Spring Joint Computer Conference, AFIPS Press, Vol. 40, 1972, pp. 617-625.

12. Champine, G.A. "Six Approaches to Distributed Databases", Datamation, Vol. 23 No. 5, Technical Publishing Company, Barrington Illinois May 1977, pp. 69-72.

13. Chu, W.W. "Optimal File Allocation in a Computer Network", in Computer Communication Networks, Kuo, F.F. editor, Prentice-Hall Computer Applications in Electrical Engineering Series, Prentice-Hall Inc., Englewood Cliffs NJ, 1973.

14. Deppe, M.E.; and Fry, J.P. "Distributed Databases: A Summary of Research", Computer Networks, Vol. 1 No. 2, North-Holland Publishing Company Amsterdam, The Netherlands, September 1976, pp. 130-138.

15. Ellis, C.A. "A Robust Algorithm for Updating Duplicate Databases", Proceedings 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California, Berkeley California, May 1977, pp. 146-158.

16. Foster, J.D. "The Development of a Concept for Distributive Processing", Proceedings Twelfth IEEE Computer Society International Conference (COMPCON 76), San Francisco California, 1976.

17. Foster, J.D. "Distributive Processing for Banking", Datamation, Vol. 22 No. 7, Technical Publishing Company, Barrington Illinois, July 1976.

18. Fry, J.P.; Frank, R.L.; and Hershey, E.S. III "A Developmental Model for Data Translation", Proceedings ACM 1972 SIGFIDET Workshop on Data Description, Access, and Control, Denver Colorado, 1972, pp. 77-105.

19. Fry, J.P.; Smith, D.P.; and Taylor, R.W. "An Approach to Stored Data Definition and Translation", Proceedings ACM 1972 SIGFIDET Workshop on Data Description, Access, and Control, Denver Colorado, 1972, pp. 13-55.

20. Fry, J.P.; Smith, D.P.; Taylor, R.W.; Frank, R.L.; Lum, V.Y.; Behymer, J.A.; and Shneiderman, B. "Stored-Data Description and Data Translation", Information Systems, Vol. 2 No. 3, 1977, pp. 95-160.

21. Hammer, M.M.; and Shipman, D.W. Technical Report in Progress, Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139, 1977.

22. Heller, J.; and Osterer, L. "The Design and Data Model of the BNL Archive and Dissemination System", Proceedings 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory University of California, Berkeley California May 1977, pp. 161-181.

23. Housel, B.C.; Lum, V.Y.; and Shu, N.C. "Architecture for an Interactive Migration System (AIMS)", Proceedings ACM 1974 SIGMOD Workshop on Data Description, Access, and Control, Ann Arbor Michigan, May 1974, pp. 157-169.

24. Housel, B.C.; Smith, D.P.; Shu, N.C.; and Lum, V.Y. "DEFINE - A Non-Procedural Data Description for Defining Information Easily", Proceedings ACM Pacific 75, San Francisco California, April 1975, pp. 62-70.

25. Johnson, P.R.; and Thomas, R.H. "The Maintenance of Duplicate Databases", Network Working Group RFC #677 NIC #31507, January 27, 1975. (Available from Network Information Center, Stanford Research Institute, Menlo Park California 94025).

26. Kleinrock, L.; and Tobagi, F. "Random Access Techniques for Data Transmission over Packet-Switched Radio Channels", AFIPS Conference Proceedings, Vol. 44, pp. 187-201, National Computer Conference, 1975.

27. Lamport, L. "Time, Clocks and Ordering of Events in a Distributed System", Massachusetts Computer Associates Report # CA-7603-2911, March 1976. Also submitted to CACM.

28. Lampson, B.; and Sturgis, H. "Crash Recovery in a Distributed Data Storage System", unpublished paper, Computer Science Laboratory, Xerox Palo Alto Research Center, Palo Alto California 94304, 1976.

29. Levin, K.D. Organizing Distributed Databases in Computer Networks, Technical Report No. 74-09-01, Department of Decision Sciences, The Wharton School, University of Pennsylvania, 1974 (PhD dissertation).

30. Levin, K.D.; and Morgan, H.L. "Optimizing Distributed Databases - A Framework for Research", Proceedings 1975 AFIPS National Computer Conference, AFIPS Press, Vol. 44, 1975, pp. 473-478.

31. Lum, V.J.; Shu, N.C.; and Housel, B.C. "A General Methodology for Data Conversion and Restructuring", IBM Journal of Research and Development, Vol. 20 No. 5, May 1976, pp. 483-497.

32. Mahmoud, S.; and Riordan, J.S. "Optimal Allocation of Resources in Distributed Information Networks", ACM Transactions on Database Systems, Vol. 1 No. 1, March 1976, pp. 66-78.

33. Marten, A.G.; and Fry, J.P. "A Data Description Language Approach to File Translation", Proceedings ACM 1974 SIGMOD Workshop on Data Description, Access, and Control, Ann Arbor Michigan, May 1974, pp. 191-205.

34. Metcalfe, R.M. Packet Communication, Technical Report No. TR-114, Laboratory for Computer Science, M.I.T., Cambridge Massachusetts, December 1973 (PhD dissertation).

35. Metcalfe, R.M.; and Boggs, D.R. "Ethernet: Distributed Packet Switching for Local Computer Networks", Technical Report, Xerox Palo Alto Research Center, Palo Alto California, 1976. (Abstract also appears in Proceedings 1976 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California, Berkeley California, May 1976, p. 238).

36. Navathe, S.B.; and Fry, J.P. "Restructuring for Large Databases: Three Levels of Abstraction", ACM Transactions on Database Systems, Vol. 1 No. 2, June 1976, pp. 138-158.

37. Pliner, M.; McGowan, L.; and Spalding, K. "A Distributed Data Management System for Real-Time Applications", Proceedings 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California, Berkeley California, May 1977, pp. 68-86.

38. Ramirez, J.A.; Rin, N.A.; and Prawnes, N.S. "Automatic Generation of Data Conversion Programs Using a Data Description Language", Proceedings ACM 1974 SIGMOD Workshop on Data Description, Access, and Control, Ann Arbor Michigan, May 1974, pp. 207-225.

39. Rosenkrantz, D.J.; Stearns, R.E.; and Lewis, P.M. "A System Level Concurrency Control for Distributed Database Systems", 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California, Berkeley California, May 1977, pp. 132-145.

40. Rothnie, J.B. "Evaluating Inter-Entry Retrieval Expressions in a Database Management System", Proceedings AFIPS 1975 National Computer Conference, AFIPS Press, Vol. 44, 1975, pp. 417-424.

41. Rothnie, J.B.; and Goodman, N. A Study of Updating In a Redundant Distributed Database Environment, Technical Report No. CCA-77-01, Computer Corporation of America, 575 Technology Square, Cambridge Massachusetts 02139, February 15, 1977.

42. Rothnie, J.B.; and Goodman, N. "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases", 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California Berkeley California, May 1977, pp. 39-57. (Also available from Computer Corporation of America, 575 Technology Square, Cambridge Massachusetts 02139, as Technical Report No. CCA-77-04).

43. Rothnie, J.B.; Goodman, N.; and Bernstein, P.A. "The Redundant Update Algorithm of SDD-1: A System for Distributed Databases (The Fully Redundant Case)", First International Conference on Computer Software and Applications (COMPSAC 77), IEEE Computer Society, Chicago Illinois, November 1977. (Also available from Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139, as Technical Report No. CCA-77-02).

44. Schneider, C.M.; and Desautels, E.J. "Design of a File Translation Language for Networks", Information Systems, Vol. 1 No. 1, January 1975, pp. 23-31.

45. Schneider, L.S. A Relational Query Compiler for Distributed Heterogeneous Databases, unpublished paper, Martin Marietta Corporation, Denver Colorado, 1977.

46. Shoshani, A. "A Logical-level Approach to Database Conversion", Proceedings ACM 1975 SIGMOD Conference on the Management of Data, San Jose California, 1975, pp. 112-122.

47. Shoshani, A.; and Brandon, K. "On the Implementation of a Logical Database Converter", Proceedings First International Conference on Very Large Databases, Framingham Massachusetts, September 1975, pp. 529-531.

48. Shu, N.C.; Housel, B.C.; and Lum, V.Y. "CONVERT: A High Level Translation Definition Language for Data Conversion", Communication of the ACM, Vol. 18 No. 10, October 1975, pp. 557-567.

49. Shu, N.C.; Housel, B.C.; Taylor, R.W.; Ghosh, S.P.; and Lum, V.Y. "EXPRESS: A Data Extraction, Processing and Restructuring System", ACM Transactions on Database Systems, Vol. 2 No. 2, June 1977, pp. 134-174.

50. Sibley, E.H.; and Taylor, R.W. "A Data Definition and Mapping Language", Communications of the ACM, Vol. 16 No. 12, December 1973, pp. 750-759.

51. Smith, D.P. "A Method for Data Translation Using the Stored Data Definition and Translation Task Group Languages", Proceedings ACM 1972 SIGFIDET Workshop on Data Description, Access, and Control, Denver Colorado, 1972, pp. 107-124.

52. Stearns, R.E.; Lewiss, P.M. II; & Rosenkrantz D.J. "Concurrency Controls for Database Systems", Proceedings of the 17th Annual Symposium on Foundations of Computer Science, IEEE 1976, pp. 19-32.

53. Stonebraker, M.; and Neuhold, E. "A Distributed Database Version of INGRES", 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California, Berkeley California, May 1977, pp. 19-36.

54. Stonebraker, M.; Wong, E.; Kreps, P.; and Held, G. "The Design and Implementation of INGRES", ACM Transactions on Database Systems, Vol. 1 No. 3, September 1976, pp. 189-222.

55. Sutherland, W.R. Distributed Computation Research at BBN, Vol. III, BBN Technical Report No. 2976, December 1974.

56. Thomas, R.H. "A Resource Sharing Executive for the Arpanet", Proceedings AFIPS National Computer Conference, AFIPS Press, Vol, 42, 1973, pp. 155-163.

57. Thomas, R.H. A Solution to the Update Problem for Multiple Copy Databases which Uses Distributed Control, BBN Report No. 3340, July 1975.

58. Whitney, V.K.M. A Study of Optimal File Assignment and Communication Network Configuration in Remote-Access Computer Message Processing and Communication Systems, PhD Dissertation, Department of Electrical Engineering and College of Engineering, University of Michigan, September 1970.

59. Wiseman, T. "Ambitious EFT: Quintet of Banks, Quartet of Vendors", Computerworld, Vol. 11 No. 23, Newton Massachusetts, June 6, 1977, pp. 1,6.

60. Wong, E. "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases", 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California, Berkeley California, May 1977, pp. 217-235. (Also available from Computer Corporation of America, 575 Technology Square, Cambridge Massachusetts 02139, as Technical Report No. CCA-77-03).

61. Wong, E.; and Youssefi, K. "Decomposition - A Strategy for Query Processing", ACM Transactions on Database Systems, Vol. 1 No. 3, September 1976, pp. 223-241.

Six Approaches to Distributed Data Bases

by G. A. Champine

The technology is here, and we know quite a bit about where to apply it.

During the past few years, the topic of distributed data base systems has received much attention, but very little information has been shared about actual implementation examples, or how these examples fit into the global picture.

The reason for the interest in distributed data base systems is because they provide a solution to some very real problems for the geographically distributed organization which needs to preserve a unified information-sharing and processing system. Then too, information processing and storage costs have been decreasing at twice the rate of decreases in data communication costs. Therefore, it is becoming increasingly attractive to substitute lower cost local processing and storage for (relatively) more expensive data communications. There are also advantages in system reliability and system performance for those applications where distributed data bases are appropriate.

The geographically distributed system user currently has two approaches available for implementing a unified data system.

1. The files can be centralized, with access by remote terminals. This approach provides for economy of scale and for easy central implementation and control. However, communication costs can be high and the system is quite vulnerable to a failure at the central site or in the communication system.

2. The system can be implemented with a number of (generally small) computers (nodes) at the various locations, each with its own mass storage and terminals. There may or may not be a central node. Applications software in the nodes can send data to, and receive data from, other nodes. With this approach, each node can access only local files, and the burden falls on the user to generate application software to do all manipulation of data at other nodes. The cost and effort to do this require a very high level of capability on the part of the users. However, there are benefits in failsoft operation, high transaction rates, and low communications cost.

A "typical" node does not exist because of the application-dependent nature of the configurations. However, a number of systems have been implemented with a (non-central) nodal configuration something like the following:

- Minicomputer with 64 to 128KB
- 10 to 20 crt terminals
- 10 to 20 MB mass storage
- 1 to 2 medium speed printers (100-300lpm)
- magnetic tape
- communications equipment (1200-2400bps)
- transaction/batch software

A number of the central nodes have consisted of medium to large scale conventional systems the size of a Univac 1100/20 or larger. However, the non-central nodes can range from small to large machines, depending on application requirements, as can the central node. Thus, it seems that the distributed systems will not obsolete either the medium or large scale systems, but instead will complement and extend their application.

Centralization vs. distribution

The following situations can be identified as typical candidate applications where a distributed data base would be more efficient than a centralized data base.

1. Large volumes of data are generated at many locations, and fast access is required. Some of the data, and summarizations of all of the data, are needed at a central site in a timely manner. An application example of this is a chain of retail stores requiring current status on operations.

2. A large amount of data is generated centrally, but fast access is required at remote locations. Either the central or remote site can update the information. An application example of this is the production scheduling of manufacturing plants. The production schedules are used by individual plants to produce parts explosions and net material requirements every day. Engineering changes can be made by each plant.

3. Remote locations generate large volumes of data for fast response to immediate inquiry. The vast majority of the inquiries are local, but fast response is needed infrequently from other locations. One example of this is a unified stock quotation system covering the five stock exchanges across the country; another example is interairline passenger reservation processing.

From a functional basis, each of these applications could be satisfied with either a centralized or distributed system. However, each of the two approaches has advantages and disadvantages.

In general, the advantage of a centralized approach are the disadvantages of a distributed approach and vice versa. They are, in broad form:

Centralized advantages/ distributed disadvantages

- operations economy
- hardware economy of scale
- unified control
- easy intrafile communication
- easy update/retrieval
- compatibility

Distributed advantages/ centralized disadvantages

- communication failsoft capability
- central site failsoft capability
- lower communications data rate and cost
- configuration flexibility
- high system performance (fast response and high transaction rate)
- modular implementation
- modular upgrade

The failsoft capability of distributed systems for communication or central node failure arises because files and records are normally distributed on the basis of record activity. The records that are most active for a given node

SIX APPROACHES

are generally stored at that node. If the central node becomes unavailable, the high activity records can still be processed. The performance requirement for the communication system may be reduced for the same reason.

At the same time, the benefits of economy of scale for large central systems have been weakened or negated by the advent of low cost small processors and memories, both main memory and mass storage. Also, there is a management trend to decentralize operations to obtain better control.

Often the application requirements are conflicting, but generally the distributed system approach would be selected if:

- Failsoft considerations are important.
- The data base can be distributed according to activity (partitioned or segmented).
  Performance requirements (transactions/second) are high.

Implementation alternatives

A generalized system diagram for a three level distributed data system is shown in Fig. 1. Each node in the system is assumed to have data storage, data processing, and data entry/retrieval capability in addition to data communication.

If the data base accessed by transactions can be segmented into local files in the same way as the nodes are geographically distributed (that is, they are congruent), then each data item is stored only once on its own system; all transactions and data are local, and each node is aware of its own files.

However, a different situation arises when transactions must reference data that cannot be local to one node; this situation may require distributed data. The following approaches to data distribution are available:

1. Store duplicate (replicated) copies of the total file at all locations, and propagate updates to all locations.

2. Store the records of the data base at the node that exhibits the highest activity for that record, and transfer transactions that cannot be handled locally to the appropriate node, either the central node or the node holding the record.

When access is made to data not held locally, the transaction is treated as an exception, and at least two options exist for its handling. The transaction can be forwarded to the next higher node where a master directory exists to forward the transaction to the proper location. Or, an updated copy of all data may exist at the next higher node so that the transaction can be handled there directly.

The task of selecting the best match between data base characteristics and distribution method requires extensive system analysis. However, on an intuitive basis, the best approach to distribution of small files is to simply replicate them and propagate updates to all nodes holding a copy (of course, appropriate synchronization methods must be used). If the file is large and has a small exception rate (say, 25%), the best approach would seem to be to store records at nodes by activity and propagate exceptions.

However, if the file size is large and the exception rate is high, it may be necessary to centralize the file. This can be summarized in the following table.

Exception rate	File size	Distribution method
—	small	replication
small	large	partition
high	large	centralize

A number of distributed data base systems have been implemented and are now operational. The following have been selected for discussion because each one represents a unique approach to implementation of distributed data base systems:

SITA	ARPANET
Celanese	Lowes, Inc.
Bank of America	Aeroquip

SITA

SITA (Société Internationale de Telecommunications Aéronautiques) provides a system for communication between airline data bases for the purpose of making passenger reservations on airplanes in Europe and Asia. Assume that a hypothetical Mr. Jones wants to make an airplane trip that requires travel on three different airlines, Northwest, Air France, and Lufthansa, and that he places his reservation with Northwest.

The Northwest reservation agent in

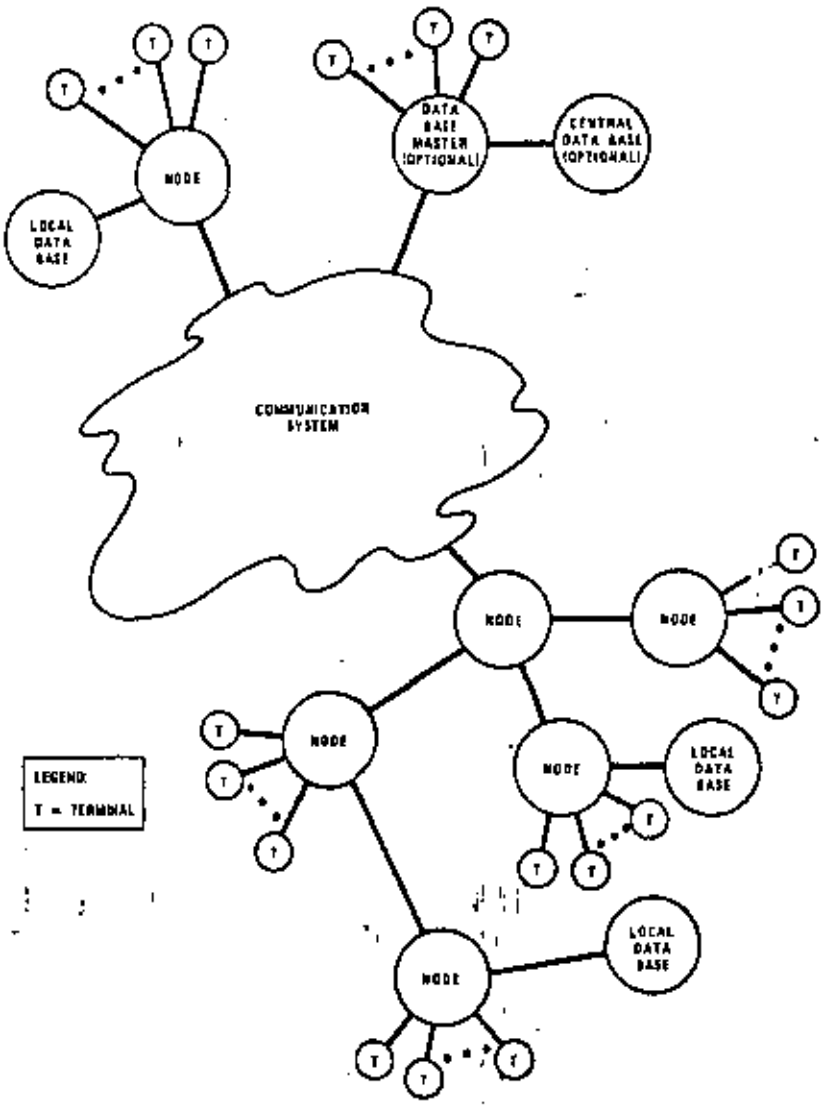


Fig. 1. The two basic approaches to distributing data bases are to replicate the entire data base at each node, which will obviously work best when the file sizes are small and communications costs are high, or to partition the data base, which works well when most transactions are to "local" large files. When most transactions are not to local files and file sizes are large, centralization of some form may still be the answer.

Minneapolis will check the (centralized) Northwest passenger data base for the appropriate flight to determine if space is available for the Northwest portion of the trip. Assuming it is, the agent will enter a transaction in the Northwest system to determine if space is available on Air France and Lufthansa. These data bases are not held by the Northwest system, so the inquiries (transactions) are forwarded to SITA for processing since Air France and Lufthansa are European airlines.

SITA determines the proper destinations for the inquiries and routes the transactions to the proper locations (Paris and Frankfort, respectively). The returning information also comes through SITA for routing back to Minneapolis.

The classification of this system is a virtual, partitioned data base system with the directory method of routing exception transactions. ARINC (Aeronautical Radio, Inc.) performs the same function for U.S. airlines that SITA performs for European/Asian airlines and in the same manner.

CELANESE

Celanese is a large U.S. textile manufacturer with a number of geographically distributed facilities with principal locations at Charlotte, S.C., and Shelby, N.C. Each location has its own computer which serves as a node in the system. An integrated data base is maintained jointly on these two nodes, as well as a number of other nodes in the system including laboratory quality control information, warehouse inventory control, and shipping information.

Each of the several nodes enters its own tranactions into the integrated data base. In addition to the usual transactions performed by the laboratories, orders (including revisions), customer information, and product descriptions are entered from multiple locations at Charlotte, while production, billing, and order shipment summary data is obtained. At Shelby, data is obtained from the production and warehouse nodes for purposes of routing products to storage space, for shipping, inventory control, production control, and for storage space control.

The method of data base distribution at Celanese is to maintain replicated files at both Charlotte and Shelby, with periodic updates of the data from both locations. In addition to the common data that is replicated, each location also has unique data.

BANK OF AMERICA

The Bank of America is a large financial institution headquartered in California, with over 1,000 branches and 11 million accounts. Data processing services are provided on a batch basis from centers in San Francisco and Los Angeles. In spite of the large amount of data on customer accounts held in the bank's (batch) data base, this data was not available to the tellers because of a lack of on-line access capability. The result was that in cash payout situations, such as check cashing and credit card cash advances, the tellers had no way to determine account status quickly.

Therefore, predetermined rules had to be established based on the amount of the transaction in an attempt to limit losses without invoking too many customer complaints about delays encountered while manually checking account status. Even so, check cashing loss often ran in excess of $3 million annually.

To improve this situation, an on-line teller information system was established with the goals of:

- a 33% reduction in check cashing losses
- a 40% reduction in interoffice phone expense
- a 16% reduction in office staff

A pilot system for 100 offices was successful in meeting these goals, and the system has now been expanded to 209 branches and 2,100 terminals on an operational basis. The distributed system is now approaching operational status.

The data base is partitioned between the two data centers, with exception transactions from one node forwarded to the other for processing. Each node is fully redundant so that processing can continue in spite of a hardware failure. The distributed data base approach is estimated to have a lower development cost by 40% than a centralized approach and is estimated to be some $4 million lower in yearly operating cost. The savings from this system are expected to exceed those operating costs by $3 million per year.

ARPANET

A software system entitled RSEXEC (Resource Sharing Executive System), has been developed by the Bolt, Beranek, and Newman Co. for use on the ARPANET to provide distributed resource sharing. A major characteristic of RSEXEC is a distributed file capability which spans host computers (nodes) and supports uniform file access and automatic maintenance of replicated files. RSEXEC is currently operational on ARPANET.

LOWES COMPANIES, INC.

Lowes Companies, Inc. is a chain of 140 retail lumber/hardware stores located in the southeastern United States. In the early 1970s, the decision was made to move to an on-line inventory control and customer invoicing system. A centralized system was first considered, with the data base located at the corporate headquarters and communication lines connecting it to terminals in all 140 stores.

This approach served the centralized operations well, such as central purchasing, warehousing, and distribution/billing to the stores. But it was not a good approach for the individual stores because the data base for each store was logically independent from all others, and large enough to benefit from economy of scale at the local level. For this and other reasons, including heavy communications cost, a distributed system was examined.

The approach finally implemented was to install a minicomputer with disc and up to 16 terminals at each store. Each store is connected to the (functionally distributed) central system by data communication lines, and inventory and sales summary information is transmitted to the central system automatically each night. The system also receives information from the central site each night, such as new price information.

System installation is now basically complete. Store personnel have immediate on-line access to inventory, pricing, and customer account information which yields a 30% increase in sales person efficiency. It also yielded intangible but definite improvements in control over credit sales, accounts receivable, and price accuracy which results in additional cost savings.

The Lowes system is, then, a partitioned data base with all files and transactions handled locally.

AEROQUIP CORPORATION

Aeroquip Corp., a subsidiary of Libby-Owens-Ford, is a manufacturer of fluid power components, including hoses, fittings, and couplings in many sizes. The company had been using an on-line query system locally at the corporate hedquarters for inventory control and order processing, and wished to extend the capability to a number of other manufacturing locations scattered from Georgia to Oregon. The objective was to provide on-line order processing, inventory control, credit checking, and shipping documents, to facilitate shipping most orders within 24 hours after receipt.

The classical centralized data base approach was examined first and found to be too expensive from a data communications standpoint.

The system ultimately selected uses intelligent terminals at each node, supported by a local data base on disc. These intelligent terminals are con-

SIX APPROACHES

nected by low performance communication lines to the central data base on a large scale mainframe, where a complete copy of the entire data base is maintained. Each node maintains a local subset of the master data base that is needed by that node.

In operation, the central site dials each node once every five minutes over a WATS line for transactions that cannot be handled locally. At night the central site dials each node in turn and obtains the accumulated transactions in compressed form and sends back updated records and output reports. The system became operational in 1973.

This approach, then, is a partitioned data base with exception transactions sent to the central node where they are handled by a complete copy of the file.

At least a partial proof

The above examples have been selected to illustrate some of the approaches to distributed data base systems. (See Table.)

It would appear that, at least in some instances, distributed data base systems have lived up to expectations. Although there is still a great deal to be learned about the proper design, implementation, and appropriate application of distributed data base systems, a basic technology and sphere of application does seem to exist.

It has been suggested by some that the advent of distributed data base systems will eliminate the need for large scale computers. The evidence does not seem to support this contention. Of the examples cited, SITA and ARPANET use large scale computers as nodes; Celanese and Aeroquip use both large and small computers as nodes; and Lowes and Bank of America use small computers as nodes. It seems more likely that, while small computers may replace large ones in some distributed data applications, they will complement them or fail to compete with them in others. Thus, we may well see a complete spectrum of alternative configurations in use as the technology matures, depending on the application.

SITA	partitioned, exception transactions handled by directory at master node
Celanese	replicated files
Bank of America	partitioned, exception transactions sent to other nodes for processing (no master)
ARPANET	this network is accessed by such a diverse user community that all design approaches may be currently being used
Lowes	segmented, all files local
Aeroquip	partitioned, exception transactions handled by duplicate data at master node

Dr. Champine is senior staff consultant at Sperry Univac, where he is responsible for managing the advanced technology program, and for technology planning of future large scale commercial computer systems. In his 19 years with the company he has held several technical and managerial positions in software and systems design, the most recent of which was as director of advanced systems design for large scale systems.

CASOS PRACTICOS DEL DISEÑO DE SISTEMAS DE INFORMACION

THE DEVELOPMENT OF DATA-BASE TECHNOLOGY

(artículo de la revista ACM COMPUTING SURVEYS)

AGOSTO, 1980

Guest Editor's Introduction • • •

The Development of Data-Base Technology

E. H. SIBLEY

Department of Information Systems Management, University of Maryland, College Park, Maryland 20742, and National Bureau of Standards, Washington, D.C. 20234

I should like to thank Elliott Organick for inviting me to serve as Guest Editor for this issue of COMPUTING SURVEYS, which has become our industry's most effective educational journal. I believe this issue deals with the most important topic in computing today: data-base technology.

Here we attempt an integrated approach to a very disoriented field. Problems abound: differences in terminology, differences in modeling, and differences in implementation confuse the potential user, who is faced with almost unanswerable questions, such as:

- Should I wait for the dust to settle, or start to use data-base technology now?; and
- If I go data-base, which type of system do I choose?

Such problems are found in any evolving technology, especially when it is associated with a fast developing industry, such as computing; while some problems appear more philosophical than real, others arise from a poor understanding of new concepts. Here we shall try to answer some of the questions and reduce the confusion. But obviously, one issue of COMPUTING SURVEYS cannot be all encompassing; data technology is a field which already boasts hundreds of articles, and textbooks by the dozen. Thus, this issue confines itself to an explanation of various models of data-base systems, showing their differences and similarities, while trying to relate the models to their implementation in current commercial and experimental systems.

We were faced with one major problem in trying to provide an integrated issue: every model uses its own terminology. We therefore decided to attempt to use both a common terminology and a single example wherever possible. This is by no means a simple task; we must not only define terminology and apply it to descriptions of various models, but also show how these terms differ from those used by others in discussing the same ideas. There is no standard, and we are forced to be arbitrary.

It was not possible to discuss every model or implementation in one issue; in fact, it is difficult to deal with anything but basic concepts. We admit to errors and omissions, and apologize for them; a real attempt was made to solicit aid from a wide variety of experts. We issue them a blanket vote of thanks and apologize for inadvertent omissions.

OUTLINE OF THE ISSUE

Figure 1 provides a graphic overview of this issue for readers. The first article, by Fry and Sibley, could be called an "entry" to the issue. Dependent on this are two articles, essentially at the same "level"; one by Chamberlin, the other by Taylor and Frank. They discuss two different and independent approaches. The article by Tsichritzis and Lochovsky, using the common terms and example of the first paper and discussing

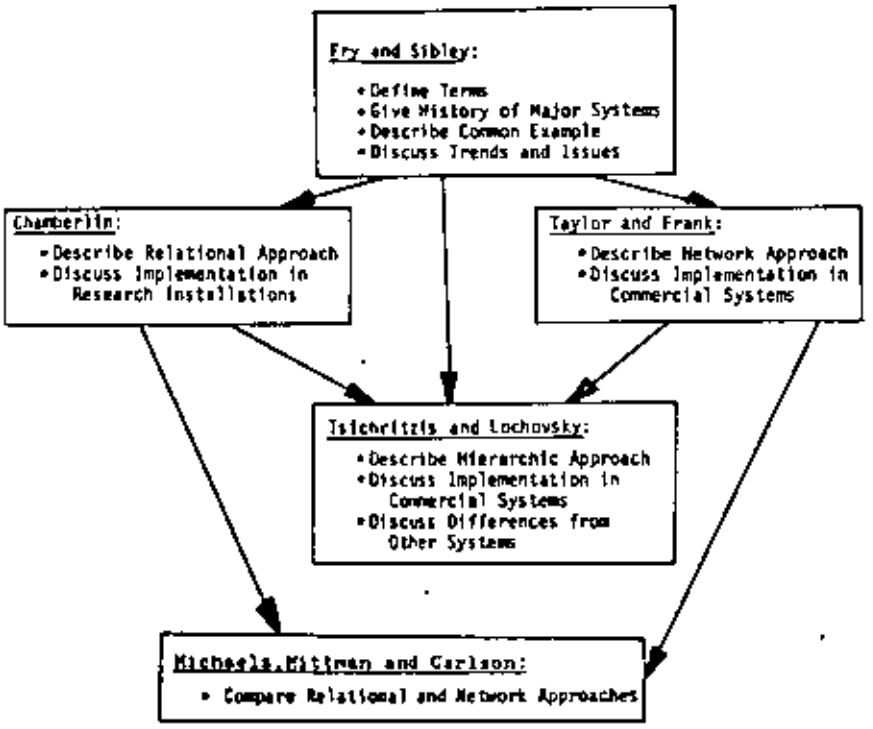


Figure 1: The structure and contents of this issue.

differences between the hierarchic and other approaches, depends on all three previous papers. Finally, the paper by Michaels, Mittman, and Carlson compares the approaches described in the second and third papers; it is consequently dependent on both of these.

PRESSURES TOWARD DATA-BASE TECHNOLOGY

Data-base technology is one of the most rapidly growing areas of computer and information science. In less than twenty years, with the greatest part of the development in the past eight years, data-base systems have come from nothing to be a major topic of current interest. Top management of major corporations have grown to appreciate the importance of their data bases; government regulatory agencies are already worrying about the implementation of privacy and freedom of information acts and their relation to data banks. This proliferation of interests started somewhere; thus, our first question is:

- *Why did it all happen?*

The pressures of the "Computer Age" arise from a fascinating new technology supplied with inexpensive equipment. The efficient, fast, accurate, and economical way a computer can perform numeric and logical operations (compared to the slow, inaccurate human counterpart) has forced automation of many previously manual operations in universities, government, and the private sector. In the early days of computers, automation merely entailed conversion from manual operations, with little attempt to integrate any resulting system. The typical data-processing operation of the fifties and sixties was created in this manner. Data* was used in the same way as before; the inefficient dupli-

* In Computing Surveys, the word *data* is used as a collective noun. Although "datum" may be correct for hard-line grammarians, it will not be found here.

cation of data and effort was continued with duplication of computerized data and procedures. The operation seemed successful: it reduced overall cost, and therefore little thought was given to further improvement of the system. In time, astute data processing managers recognized a problem: while stored within the data vaults of the organization (probably in the form of bits on a tape), data was essentially unavailable. The programs that input, stored, and used the data were essentially the owners of the data. Any other user found it difficult to obtain, integrate, or transform the "available" data for use in another program. Thus, every new need for data involved writing a new program to obtain the data before it could be processed by yet another program, and even this was difficult—the data formats were "locked" in the original programs, and sometimes the original object code had been lost!

This essential unavailability of otherwise transferable data gave rise to the question:

- *Why not integrate the data?*

This led to the thought that integration, were it possible, could be achieved by defining the data format, storing it as a "data definition," and allowing general-purpose "data-base management" software to access it. And this gave rise to the prime concept of the generalized data-base management system.

In dealing with a system to store and access data for a set of different programs, and consequently for a set of different types of users, two further questions arose. First:

- *Can we access this data through our current computer languages?*

This involves either specification of additional commands in conventional programming languages, or the provision of calls to special subroutines which allow access to the data base. And second:

- *Why not allow a higher-level language for ad hoc use of the data base?*

This can be achieved by providing a special query language as an interface. While inefficient, the first prototype systems appeared very successful to users, and the first commercial systems started to appear. Then the first problems arose.

DISADVANTAGES OF DATA INTEGRATION

The industry congratulated itself on reducing data redundancy and improving its availability, but it also introduced the potential for disaster. The first problem with integration arises because the data base is now more vulnerable to destruction through machine malfunction, personal error, or deliberate human tampering. The loss of "quality" in a data base (including total destruction) by any of these means may be considered a threat to the organization, because data is one of its most valuable assets. "Integrity" techniques are therefore a necessity.

The other threat to the integrated data base relates to its "security" and accuracy. Most enterprises have some secret processes or private material which should be protected against theft or access by unauthorized people. To achieve this protection, the system must ensure that it is *secure*, i.e., that any information which is private to the organization is safe from unauthorized dissemination or tampering. The problem with erroneous information is that it might result in an incorrect (adverse) decision about a person or enterprise. Government and other enterprises are vitally interested in both of these aspects, which have been termed "information privacy." Traditionally, *privacy* has been defined as the right of an individual or organization to be "left alone"—it usually is considered the right to retain certain non-public information without threat of disclosure. An integrated data base threatens privacy: it becomes easier to collect, and to unwittingly divulge information of a confidential nature (which may have been legally obtained from the individual or enterprise) to some other unauthorized person or agency. Today, information privacy has been expanded to include the right of the individual or organization to know what "personal" information is retained on any data base. It also implies that the individual or enterprise has the right to challenge the data, causing either its correction, or the addition of a statement that the fact is under dispute. As an example: a person may wish to ensure that data given

confidentially to an agency for the purpose of obtaining a credit card is not divulged capriciously to a neighbor; the person may also wish to know what other information is on file at a credit bureau, and be able to correct or dispute any potentially threatening fact.

When data is integrated and readily available through either program or ad hoc query interfaces to a community of users, the possibility of loss of integrity, and the ability to penetrate the security (thereby threatening privacy) obviously increases. Early data-base management systems often needed to be retrofitted to ensure reasonable integrity and security.

The placing of controls within the system brought another new issue into view—the question of who was to make the policy decisions, who would issue passwords and who was really allowed to make decisions on data formats. A new post was needed.

THE DATA ADMINISTRATION CONCEPT

Data administration is really a special form of managerial control which includes both authority over data integrity and security, and responsibility for overall efficiency. Because the data base for the enterprise was growing in total size and complexity, new measures for improving efficiency were possible. The question arose:

- *Should programmers be allowed to define their own data?*

The implementation of central authority for data definition made it impossible to allow the programmer this flexibility. By introducing a single data authority, and by providing information about the community of users, the "best" data structure could be defined: this data structure is efficient for the community of users rather than for any one particular user.

As a consequence of the advent of a new technology and new managerial control mechanisms, the computing operation started to take on a new importance. Government and business agencies found themselves with expensive data-processing operations which were being run by relatively low-level management. Some commercial organizations found that equipment requiring presidential (or even board) approval, was being run by managers at the fourth or fifth level of pecking order. This had distorted the organization, and it has already led to organizational changes: in fact, we now see "Vice President of Information Systems"—a far cry from the lowly "Data Processing Operations Manager" of yesterday.

DATA BASE VERSUS DATA PROCESSING

We have seen that data which had previously been duplicated and spread throughout the organization is now being drawn into a unified and sometimes monolithic system. This represents the concentration of a valuable asset. In the past, the only major identifiable, tangible asset of a corporation was its money, or objects immediately convertible to money. Accountants had learned how to audit and control the flow of money. The advent of data management suggested the possibility of the audit and control of data.

In exactly the same way that an accountant determines the accuracy of money flow, a data-base management auditor could determine accuracy, quality, and privacy aspects of the data. This gives rise to the concept of data auditing. New and pending state and federal government regulations on privacy and freedom of information would make the enterprise legally accountable for its data. Consequently, the matter of ad hoc use and poor control over the application and dissemination of information is suddenly a real concern. Someday soon a high-level administrator will be sentenced (fined and maybe even given a prison term) for contravening these regulations—and then the entire industry will tighten its controls—almost overnight. Thus three factors have combined to imply a need for more effective and automated control mechanisms to be built into the automated systems, with reasonable safeguards against unauthorized access. The three motivating factors are:

- the regulations of government to insure privacy;
- the aspirations of management for more effective control over its operations; and
- the understanding of auditors of the need to retain quality in all financial and nonfinancial data.

The data-base management system today

allows data to be shared by a community of users, while insuring the integrity of the data over time, and providing security against unauthorized access. At the same time, the administrator may select different methods of storing the data (termed: the physical structure) for different parts of the data base, thereby aiding in providing storage and retrieval efficiency. However, to the ultimate user or application programmer this choice of physical structure should be invisible (termed: data independence), because the user should not have to worry about these internal details. These, then, are the principal objectives of a data-base management system.

Evolution of Data-Base Management Systems*

JAMES P. FRY

Graduate School of Business Administration, University of Michigan, Ann Arbor, Michigan 48109

EDGAR H. SIBLEY

Department of Information Systems Management, University of Maryland, College Park, Maryland 20742, and National Bureau of Standards, Washington, D.C. 20234

This paper deals with the history and definitions common to data-base technology. It delimits the objectives of data-base management systems, discusses important concepts, and defines terminology for use by other papers in this issue, traces the development of data-base systems methodology, gives a uniform example, and presents some trends and issues.

Keywords and Phrases: data base, data-base management, data definition, data manipulation, generalized processing, data model, data independence, distributed data base, data-base machines, data dictionary

CR Categories: 3.51, 4.33, 4.34

1. GENERALIZED PROCESSING

A data-base management system (DBMS) is a generalized tool for manipulating large data bases; it is made available through special software for the interrogation, maintenance, and analysis of data. Its interfaces generally provide a broad range of language to aid all users—from clerk to data administrator.

DBMS technology can be traced back to the late fifties, when authors such as McGee [G1 and G2][1] discussed the success of "generalized" routines. These routines were capable of sorting any file regardless of its data content (the user merely supplying parameters to direct the major elements of the sorting process); those authors then proposed that these ideas be extended into other data-processing areas, such as file maintenance and report generation. This *generalized processing* entails the building of special data functions which perform frequently used, common, and repetitive data-processing tasks. But such generality cannot be accomplished without cost. The price of generalized processing is a reduction in operating efficiency, often through interpretive processing, or a necessary increase in resources such as hardware capacity. The success of generalized processing (and consequently of generalized data-base technology) thus becomes an issue of cost tradeoff.

Hardware improvements developed over the past two decades have effected significant decreases in price/performance ratio, thereby tending to offset operational inefficiency and to emphasize the cost of application and software development. The

* This work is sponsored in part by the National Science Foundation Grant GJ 41831.

[1] *Editor's Note:* See page 35 for the key to the classification system used for references cited in this paper.

CONTENTS

1. GENERALIZED PROCESSING
2. OBJECTIVES OF DATA-BASE MANAGEMENT
   Data Availability
   Data Quality
   Privacy and Security
   Management Control
   Data Independence
3. FUNDAMENTAL CONCEPTS AND DEFINITIONS
   Elements of Logical Structure
   Introduction to Data Models
   Mapping from the Logical to the Physical Structure
4. HISTORICAL PERSPECTIVE
   Evolution of Data Definition Languages
   Centralized Data Definition: Fifties and Sixties
   Stored-Data Definition Languages: 1970's
   Development of Report Generator Systems
   Hanford/RPG Family
5. DEVELOPMENT OF DBMS
   Early Developments, Prior to 1964
   Establishment of Families: 1964–1968
   Postley/Mark IV Family
   Bachman/IDS Family
   Formatted File/GIS Family
   Vendor/CODASYL Developments: 1968 to the Present
   CODASYL/DBTG Family
   IMS Family
   Inverted File Family
   Additional Vendor Developments
6. THE PRESIDENTIAL DATA BASE EXAMPLE
7. TRENDS AND ISSUES
   Ad Hoc versus Programming Systems
   Geographically Distributed Systems
   Data-Base Machines
   To Standardize or Not?
ACKNOWLEDGMENT
CLASSIFICATION OF REFERENCES
REFERENCES
AVAILABILITY OF REFERENCES

benefits of a generalized approach can thus be summarized as the elimination of program duplication (frequently found in computing systems), and the amortization of the one-time development costs over many applications of the program.

In cases where a particular data-processing application cannot be parameterized, the usual recourse is to develop high-level languages, which are themselves a form of parameterized generalized processing, albeit with very special parameters. For example, the development of high-level interrogation languages for ad hoc requests has broadened the user access to data by providing a simple and, it is hoped, easy-to-use interface. Such an approach allows the inquirer to use a language similar to everyday English, rather than requiring him to write a program in an artificial language. Generalized data-processing techniques have evolved into a class of sophisticated, generalized software systems, one of which is the data-base management system. The reader should carefully distinguish between the terms DBMS and "data management." The latter has been used by the government to designate an administrative function, by some hardware vendors to designate their access methods, and by some software vendors to designate and embellish comprehensive packaged systems.

2. OBJECTIVES OF DATA-BASE MANAGEMENT

The Guest Editor's Introduction to this issue of COMPUTING SURVEYS discussed the concepts of data-base technology and introduced some of its objectives:

- to make an integrated collection of data available to a wide variety of users;
- to provide for quality and integrity of the data;
- to insure retention of privacy through security measures within the system; and
- to allow centralized control of the data base, which is necessary for efficient data administration.

To this we add the objective of "data independence," a term to be defined later [see page 12] in this paper. This section will deal with each of the stated objectives, relating them to the overall functional architecture of the DBMS.

While various "views of data" (the principal topic of this issue of COMPUTING SURVEYS) are important to the user interface, the requirements for quality, integrity, security, and control have far-reaching effects on the overall cost, accessibility, and per-

formance of the system. Although it is possible to add functional capabilities to an existing system, the cost of retrofitting is often prohibitive, and the post-design addition may adversely affect the system performance. Although quality, security, and control factors are given relatively scant treatment in other papers in this issue of SURVEYS, it should not be inferred that these are unimportant. In fact, the consequences of excellent or poor satisfaction of these needs may make or break a working system.

Data Availability

Everest [G12] states that the major objective of a DBMS is to make data sharing possible. This implies that the data base as well as programs, processes, and simulation models are available to a wide range of users, from the chief executive to the foreman (Everest and Sibley [G8]). Such sharing of data reduces its average cost because the community pays for the data, while individual users pay only for their share. However, under these circumstances the data cannot "belong" to any individual, program, or department; rather, it belongs to the organization as a whole.

What, then, is the overall cost of data? One way to answer this question is by observing data entry. Keypunching and verifying, or other types of data entry involving human keystroking, tend to cost about 50¢ per thousand characters input. Thus, if the average-sized file is two million characters (a figure representative of much of today's industry and government), it costs $1000 to input each average-sized file. Under certain conditions the cost of collecting data could be substantially higher, e.g., when the data must be collected by telemetry, or in long and complicated experiments.

Another expense is associated with the *lack* of data, the so-called "lost opportunity cost." If data is not available when an important decision is to be made, or if duplicate but irreconcilable data exists, an ad hoc and possibly wrong decision results. Nolan [A4] gives a scenario of a typical business where a manager knew that data existed, but some of it had been produced on a different machine and some had incompatible formats (different structures on different tapes). Moreover, none of the data definitions were easily available. The manager who needed the data for important predictions was unable to obtain answers in a reasonable amount of time.

There are two important mechanisms for making data available: the "data definition" and the "data dictionary." A data definition is a more sophisticated version of a DATA DIVISION in COBOL, or a FORMAT statement in FORTRAN; however, a data definition is supplied *outside* the user program or query and must be attached to it in some way. The *data definition* (as specified by a data administrator) generally consists of a statement of the names of elements, their properties (such as character or numerical type), and their relationship to other elements (including complex groupings) which make up the data base. The data definition of a specific data base is often called a *schema*.

When the data definition function is centralized (which is necessary to achieve the objectives of DBMS), control of the data-base schema is shifted from the programmer to the data administrator [A1]. The programmer or the ad hoc user of a query language is no longer able to control many of the physical and logical relationships. While this restricts the programmer to some extent, it means that all programs use the same definition; thus any new program can retrieve or update data as easily as any other. Furthermore, greater data definition capabilities are provided, the storage and retrieval mechanisms are hidden from the program, the formats cannot be lost, and the programmer's task is simpler.

Centralized data definition facilitates the control of data duplication, which generally entails some storage inefficiency. However, not all duplication of data is bad; a controled duplication may be necessary to allow special classes of users to obtain especially fast responses without penalizing quality for other users.

The data definition facility is inherent to all DBMS. Without it, the data base is owned by its programs, difficult to share,

and generally impossible to control. This, then, is the cornerstone of data-base management systems.

Whereas the data definition facility is the data administrator's control point, the *data dictionary* [D1] provides the means of broadcasting definitions to the user community. The data dictionary is the version of the data definition that is readable by humans. It provides a narrative explanation of the meaning of the data name, its format, etc., thus giving the user a precise definition of terms; e.g., the name TODAYS-DATE may be defined narratively and stated to be stored in ANSI standard format as Year: Month: Day.

Within the past five years a number of data dictionary packages have appeared on the market [D2]. Some of these are an integral part of the data definition function, while others provide an interface to multiple DBMS, and still others are stand-alone packages.

The dictionary will normally perform some, if not all, of the following functions: storage of the definition, response to interrogation, generation of data definition for the DMBS, maintenance of statistics on use, generation of procedures for data validation, and aid in security enforcement. Obviously, storage of the data definitions in the dictionary is obligatory.

The dictionary will normally be able to either provide formatted dictionaries (on request) or respond to a simple query for a data entry, or to do both. This facility allows ad hoc users to browse through the definitions (on- or off-line) to determine correct data names.

In some dictionary systems, especially those that augment a DBMS, the data administrator can invoke a data definition generator. This allows the administrator to pick names of elements from the dictionary, group them, and then produce a new data definition.

The dictionary may be both a collector for, and a repository of statistics on DBMS usage. These statistics can be utilized to improve the efficiency of the DBMS by regrouping elements for better accessing.

The dictionary may contain information on techniques for validation of particular elements, and the data validation statements can be used to generate procedures for input editing or other quality checking.

The data dictionary is extremely important as part of the DBMS security mechanism. If an adversary knows you are gathering data, that adversary has already violated your security. For this reason, the data dictionary should be as secure as the DBMS. Furthermore, if security requirements are retained in the dictionary they can be automatically checked (and special procedures can be invoked) every time a data definition is produced for the DBMS. This would improve security monitoring.

Data Quality

Perhaps the most neglected objective of DBMS is the maintenance of quality. Problems relating to the quality of data and the integrity of systems and data go hand-in-hand. Data may have poor quality because it was:

- never any good (GIGO—garbage in, garbage out);
- altered by human error;
- altered by a program with a bug;
- altered by a machine error; or
- destroyed by a major catastrophe (e.g., a mechanical failure of a disk).

Maintenance of quality involves the detection of error, determination of how the error occurred (with preventive action to avoid repetition of the error), and correction of the erroneous data. These operations entail precautionary measures and additional software functions within the data-base management system. The prevention and correction of the five listed causes of error will now be briefly discussed.

In dealing with normal data-processing applications, the programmer is faced with a great deal of input validation. A survey by the authors showed that about 40% of the PROCEDURE divisions of present-day industrial COBOL programs consists of error-checking statements. If the validation requirements can be defined at data definition time, then error checks may be applied automatically by the system at input, update, manipulation, or output of data, depending on the needs specified by the data adminis-

trator. Many current DBMS allow validation. Some have a check mechanism which ensures that the values conform to the stated PICTURE (like COBOL); they also check that the value is within the defined range, or that it is one of a predefined set. If a system is to support such techniques it must have special clauses in the data definition language (DDL), as well as a series of procedures to be invoked on error detection.

A second cause of poor data is human or program error. Little can be done to prevent such errors unless they contravene some validation rule, and their discovery normally involves human knowledge. The cause, however, may be detected by referring to the "audit trail." An *audit trail* is a log in some journal of all changes made to the data base. When a change is to be made, there are two important objects: the original data and the changed data. When logged, these objects are termed "before" and "after" images. Generally, these images contain the data, time, and name of the procedure causing the change. They may also record the name of the person who initiates the procedure. A *quality audit* is an attempt to determine, through examination of the before and after images, who or what procedure changed the data value. A quality audit may find that some user promulgates many errors, whereupon the data administrator may request that the user take more care (or a course in better techniques). If, however, the error appears to have been generated by some operational program, a programmer may be called in to debug it.

Sometimes an error will be detected after a procedure is partially completed. In this case, as well as when a user makes a mistake, it is often necessary to "back-out" the procedure or "back-up" the data base. This is a process of reinstating the data that has been incorrectly updated. Many data-base management systems provide an automatic facility for reinstatement, achieved by reading the before images from the audit trail and replacing any updated data with its prechange value.

Poor quality data can also be generated by an unpredicted disaster. The process of recovering from a permanent hardware failure, or of restarting after a minor error generally involves the use of the audit trail. In modern operating systems a *restart* facility is often provided. Normally, in order to restart the DBMS after a failure which does not involve physical damage to the storage devices, a "checkpoint" facility is used. A *checkpoint* is a snapshot of the entire machine condition (CPU, memory, etc.) recorded on the log. This entry presents a known condition of the entire system.

A checkpoint may be either taken by the computer operator or automatically initiated by the DBMS. Usually the latter method is triggered by a procedure which keeps count of the number of transactions processed and then initiates the checkpoint when a predefined value is exceeded. The problem with such facilities is that they often need a quiescent system, i.e., one in which the transactions are being held in queues or have been completed. This "freeze" operation may take some time. Unstarted procedures are held until the checkpoint process has been completed, causing a delay which can lead to dissatisfied users.

After any major error it is possible to back-up to the latest checkpoint on the log and then move forward along the log, replacing updated (after) images of completed transactions or reinitiating unfinished transactions. Recovery can be a complicated process, and many current data-base management systems rely substantially on the capabilities of the underlying operating system to perform the function.

Sometimes a major storage failure (e.g., a disk crash) requires replacement of hardware and total reloading of the (possibly very large) data base. It is not unusual to find commercial and governmental data bases with over one billion characters. A sequential load of such a large data base may take two to six hours on present-day computers. The reload is from a *data-base dump*, that is, from a copy taken at some time in the past (assuming possible failure of the original). A data-base dump only represents the status of the data base at a certain time, and any updating performed subsequent to that time must be replicated by using the log. Many current systems use such techniques, although some still rely on reinitiating and

reprocessing the logged update transactions. This procedure tends to be very slow.

The quality and integrity of data depend on input-validation techniques in the original data definition, logging of data-base changes, periodic snapshots of the entire machine status, and total or incremental data-base dumping. These operations require additional software in the data-base management system, both for initiation of the protective feature and for its utilization to reconstitute a good data base. However, they entail an overhead expense which adds to the normal running cost.

Privacy and Security

The third major objective of data-base management systems is privacy—the need to protect the data base from inadvertent access or unauthorized disclosure. Privacy is generally achieved through some *security* mechanism, such as passwords or privacy keys. However, problems worsen when control of the system is decentralized, e.g., in distributed data bases, where the flow of data may overstep local jurisdictions or cross state lines.

Who has the responsibility for the privacy of transmitted data? When data requested by someone with a "need to know" is put into a nonsecure data base and subsequently disseminated, privacy has been violated. One solution to this problem is to pass the privacy requirements along with the data, which is an expensive, but necessary addition. The receiving system must then retain and enforce the original privacy requirements.

Security audits, another application of the audit trail, are achieved by logging access (by people and programs) to any secure information. These mechanisms allow a security officer to determine *who* has been accessing *what* data *under what conditions*, thereby monitoring possible leakage and preventing any threat to privacy. Much of this technology is, however, still in its infancy.

Management Control

The need for management control is central to the objectives of data-base management. It includes the establishment of the data administration function and the design of effective data bases. Data administration currently uses primitive tools; a discussion of them would be beyond the scope of this paper (see [A1, 2, and 3]). However, it is important to note that data-base design involves tradeoffs, because users may have quite incompatible requirements. As an example, one group may require very rapid response to ad hoc requests, while another requires long and complicated updating with good security and quality control of the data. The implementation of a system responsive to the first need may suggest a storage technique quite different from that needed by the second. The only way to resolve such a conflict is to determine which user has the major need. If the requirements are equally important, a duplicate data base may be necessary—one for each class of user.

Although the installation of a data-base management system is an important step toward effective management control, today's data administrator faces a challenge: the available tools are simplistic and seldom highly effective. They involve simulation, data gathering, and selection techniques. Some new analytical methods appear promising [G3]. These methods select the "best" among several options of storage techniques, but they are usually associated with one particular DBMS rather than with several.

Data Independence

Many definitions have been offered for the term data independence, and the reader should be aware that it is often used ambiguously to define two different concepts. But first, we must define other terms. A *physical structure*[†] describes the way data values are stored within the system. Thus pointers, character representation, floating-point and integer representation, ones- or

[†] The terms *data structure* and *storage structure*, which were promulgated by the CODASYL Systems Committee [U2] can be attributed to D'Imperio [D1L2]. However, in computer science, the term data structure is more closely associated with physical implementation techniques such as linked lists, stacks, ring structures, etc. To prevent ambiguity we opt for the more basic terms, logical and physical structure. .

twos-complement, or sign-magnitude representation of negative integers, record blocking, and access-method are all things associated with the physical structure. The term *logical structure* describes the user's view of data. Thus, a COBOL DATA DIVISION is (mainly) a statement of logical structure; it deals with named elements and their relationships rather than with the physical implementation. A record in a COBOL program is manipulated without knowledge of the computer hardware or its access method. As an example, the data item named AUTHOR may have values FRY, SIBLEY, FRANK, TAYLOR, CHAMBERLIN, etc. Whereas the name AUTHOR is a logical phenomenon, the representation of authors is a physical phenomenon.

In the early days of DBMS, the term "physical data independence" was used. A system was said to be (physically) data independent if it could deal with different physical structures and/or different access methods, and if the user of the data base did not have to provide information on details of the structure. Thus a definition of *physical data independence* is:

> *A system is data independent if the program or ad hoc requests are relatively independent of the storage or access methods.*

Systems with physical data independence provide a discrete number of choices for implementing the physical storage of data. Other systems also allow the user to make requests with little knowledge of the logical structure of the data. Such systems, which are said to have logical data independence, may operate correctly even though the logical structure is, within reason, altered. A definition of *logical data independence* is:

> *The ability to make logical change to the data base without significantly affecting the programs which access it.*

Logical data independence has two important aspects; first, the capability of a data-base management system to support various (system or user) views of the data base, and second, the capability of the data-base management system to allow modification of these views without adversely impacting the integrity of existing applications. The latter capability is important in the restructuring function [G13], but this definition of data independence is perhaps too broad. It suggests that substantial logical change could be made without creating a need to change the programs—a difficult, if not impossible task. However, a serious attempt is being made to understand how much logical change can be made without adverse affect on the program. Some of the different models discussed in this issue of SURVEYS claim to be more data independent than others. Full data independence appears, however, to involve an understanding of *data semantics*, the formalization of the meaning of data. Research on data semantics is currently in its infancy.

3. FUNDAMENTAL CONCEPTS AND DEFINITIONS

Some important ideas were introduced when we discussed the basic objectives of DBMS. This section presents further concepts and definitions.

Unfortunately, our language is rich in its words and semantics about data. Entity, item, name, element, value, instance, and occurrence (to name a few) come ready-equipped with meaning, yet they are used in different ways. We must be precise, and are thus forced to make exact definitions for these words which we must use consistently.

Elements of Logical Structure

The starting point is to define the object of the discourse, the entity, and the process of its definition, which is a modeling process. A human being is constantly "modeling" information—a baby sees an animal and says "dog" (though it may be a horse). The process of modeling information as data often involves trial-and-error. First, information needs are determined, next data (and processes) are structured to satisfy the needs, and then data is restructured because of changes in the need or necessary improvements to the model.

The principal construct in the data structuring process is the entity:

> An information system deals with objects and events in the real world that are of interest. These real objects and events, called *entities*, are represented in the system by data. Information about

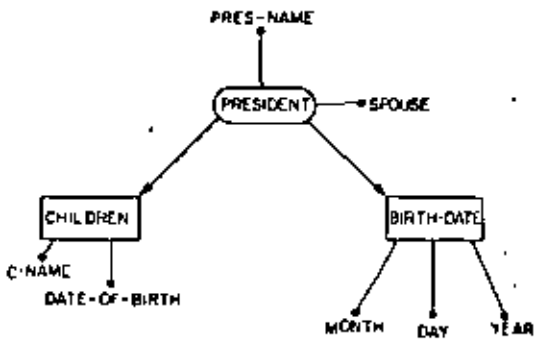


FIGURE 1. The PRESIDENT entity.

a particular entity is in the form of "values" which describe quantitatively and/or qualitatively a set of attributes that have significance in the system [M1].

Thus the goal of the data structuring process involves the collection of *data* (a set of facts capable of being recorded) about some identifiable entities that convey *information* (i.e., meaning to humans). The repository for the data is the *data base*. A data base is described in terms of its logical structure; this provides a template for the data instances which constitute the data base.

Data about an entity is generally recorded in terms of some attribute(s) that describes the entity. In the description of the data base, separate attributes are called *elementary items* or, in brief, *items*, while the collection of elementary items is termed a *repeating group* or *group*. For example, the entity PRESIDENT may be described in terms of items PRES-NAME, SPOUSE, the group BIRTH-DATE, and the repeating group CHILDREN. The group BIRTH-DATE is made up of MONTH, DAY, and YEAR, while the repeating group CHILDREN is made up of C-NAME and DATE OF-BIRTH. There may be zero or many repetitions of the CHILDREN group. The definition of the PRESIDENT entity is illustrated in Figure 1. This represents, of course, only one possible model of a definition of the PRESIDENT entity. Another user may have a different viewpoint, and need to add PARTY and STATE to the model. Thus the data base depends on the (assumed) usage, and the model may need to be changed during the life of the system.

It is possible to describe a "data model" of an entity in a formal fashion using a set-theoretic notation where:

- all repeating groups are enclosed in | | to represent the fact that they may repeat as many times as necessary, and
- all ordered sets or n-tuples are enclosed in ( ) to show that the order of the items is important.

In this way, the entity PRESIDENT may be defined as in Display 1 below.

An *instance* or *occurrence* of an entity is a set of values for each of the items of the entity definition. Any repeating group in an entity has an occurrence which consists of one value for each of its items. However, there may be potentially zero or an unlimited number of occurrences of a repeating group in the occurence of the entity. Naturally, each element value should be valid, i.e., it should conform to the rules and must be one of the possible members of the set of allowable values.

If the names of the presidents of the United States are the value set (or range) of the domain named PRES-NAME, then we have the value set in Display 2 below and can construct one instance of PRESIDENT as:

(FORD, BETTY, (7, 14, 1913),
(SUSAN, (13, 10, 57)), (JOHN, (8, 03, 52)),
(STEPHEN, (2, 21, 55)), (MICHAEL, (9, 17, 50)))).

For almost any real-world situation there are many entities of interest which are related in some fashion. In a Presidential Information System there will be entities such as PRESIDENT, CONGRESS, ELECTION, STATE, and ADMINISTRATION, all of which are interrelated; for example,

Display 1:

PRESIDENT = (PRES-NAME, SPOUSE, BIRTH-DATE, |CHILDREN|)
where BIRTH-DATE = (MONTH, DAY, YEAR)
and CHILDREN = (C-NAME, DATE-OF-BIRTH)

Display 2: Value Set of PRES-NAME = {FORD, NIXON, JOHNSON, KENNEDY, . . .}.

PRESIDENTs "WIN" ELECTIONs, STATEs are admitted during a President's ADMINISTRATION, and PRESIDENT(s) serve with CONGRESS(es). A relationship may therefore exist between the instances of two entities. Typically, there are at least three types of relationships:

- *One-to-one:* some of the Presidents are first native sons of some states; for example, Washington (one President) was the first native son of Virginia (one state).
- *One-to-many:* during an Administration several STATES may be admitted, but a state is not admitted more than once in different Administrations.
- *Many-to-many:* a President serves with many Congresses, and a Congress may serve under many PRESIDENTs.

More on this topic is discussed in other articles in this issue of SURVEYS, but before going further, the reader should note that the following statements have different meanings.

- "The relationship named A exists between two entities, B and C"; and
- "Two instances P_i and Q_i of entities P and Q, respectively, are related by A".

The first is a logical statement. It states logical relationships that may occur between two entities; for example, Presidents (B) win (won) (A) Elections (C). The second statement refers to current values of data in the data base; for example, NIXON (Q_i), a PRESIDENT (Q), wins (won) (A) the 1972(P_i) ELECTION (P).

Relationships may be explicit or implicit. The entities may be joined by some naming convention (such as WIN), or the relationship may be implied (as in the example of PRESIDENT with the repeating group CHILDREN).

Generally, the instances of certain items in a group are in one-to-one correspondence to an instance of the entity. For example, the year of an election may uniquely identify a presidential election, or the congress number may uniquely define a Congress. These items are called *identifiers* or *candidate keys*. A key may be considered either a logical or a physical phenomenon: the key may be used to identify an entity (logical), or it may cause the system to sort the set of instances of entities into an order based on the value of the key (physical). In this issue of SURVEYS, key will be considered a logical concept, but note that this definition allows "sort by key" as a physical attribute of the data base.

The discussion of entities, items, and groups has involved logical structure. The definition of this structure (a schema) requires some formal language, which is termed a *data definition language* (DDL). This language may be formatted, like a COBOL DATA DIVISION, or be relatively free-form. The following three articles in this Special Issue give specific examples of DDL usage.

Introduction to Data Models

The evolving field of data models is often hotly debated. Proponents of each model point out its advantages, but so far there is no concensus as to the best version. In reality, there is a spectrum of data models ranging from the COBOL-like "flat file" (single entity model) to the complex extended-set model.

Since COBOL, the most widely used language today, has a DATA DIVISION with data definition capabilities, it represents a good starting point for the discussion of data models. Though limited, this data definition capability allows the group (termed a RECORD in COBOL) to be defined as an 01 level, followed by the items, groups, and repeating groups at other levels. The PRESIDENT entity, discussed previously, is shown in Figure 2.

In COBOL, each item is formatted by de-

```
01 PRESIDENT.
   02 PRES-NAME PICTURE  X(20)...
   02 SPOUSE PIC X(10)...
   02 BIRTH-DATE...
      03 MONTH...
      03 DAY PIC...
      03 YEAR...
   02 CHILDREN OCCURS 0 TO N TIMES...
      03 C-NAME...
      03 DATE-OF-BIRTH...
```

FIGURE 2. A COBOL-like definition for PRESIDENT.

fining a PICTURE. Thus PRES-NAME is shown as 20 characters long, SPOUSE is 10 characters long, while DAY is two numerics.

The COBOL definition deals with one entity (defined at the 01 level), but a COBOL structure may also be termed "contained" because the groups BIRTH-DATE and CHILDREN are contained within the PRESIDENT entity (see Figure 3). There may be many levels of containment of groups within groups. There is no semantic reason why groups shown as contained in the PRESIDENT entity should not, by some other model or user, be considered separate entities; i.e., BIRTH-DATE and CHILDREN might each be entities. The relationship between PRESIDENT, BIRTH-DATE, and CHILDREN entities may, however, be constrained because the two latter are contained entities that are not really separate, but rather, are "owned" by the PRESIDENT entity. Such a model is said to be "hierarchical." Thus a *hierarchy* of entities involves a superior entity and one or more inferior entities, each of which may participate as superior entities at a third level, etc. A hierarchy represents a "tree," "bush," or "fan-out" of entities all related by a family-tree-like relationship (with no sons shared by different fathers). The topmost level of the hierarchy is termed the *entry* or *root*—terms arising because the "way in" to the entity is its entry, or (when stood on its head) it is the "root" of the tree.

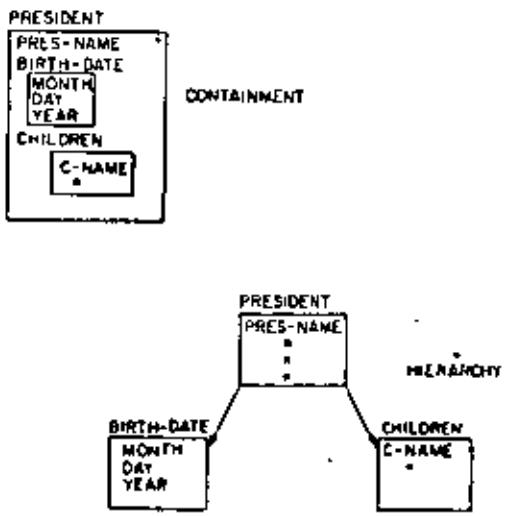


FIGURE 3. The PRESIDENT entity as contained and hierarchical structures.

Logically, containment and hierarchical representations are equivalent; however, the physical implementation of such systems causes differences in the way they are manipulated. (Hierarchic systems are discussed in this issue by Tsichritzis and Lochovsky [see page 105].) The hierarchic model has a 1 to N relationship between an owner and member entity; e.g., for one PRESIDENT there may be many (or no) CHILDREN. It also has two constraints: no member entity can be shared by the owner entities, and no entity at a lower level may own a member at a higher level in the hierarchy (assuming the words "lower" and "higher" refer to the position down a page, with the root at the top). The second constraint really follows from the first, but it has important effects.

If we first relax the multiple ownership constraint, it is possible to have the same member entity participating in two different relationships with a single owner entity. This requires a means of distinguishing the relationships. As an example, the relationships between PRESIDENT and STATE may be both ADMITTED-DURING-ADMINISTRATION and NATIVE-SON: a President may be in office when one or more STATE(s) are admitted, and one state may have zero or several native sons as Presidents. This problem can be resolved by labeling the ares (showing the relationships between the entities) with the name of the relationship, as shown in Figure 4.

The second constraint must be relaxed carefully. Most *graphical* or *network* models still retain one constraint: that no entity may participate as both owner and member in the same relationship. This may appear unfortunate or unnecessary; after all, PEOPLE do EMPLOY other PEOPLE, and some PEOPLE are PARENTS-OF other PEOPLE. However, by careful design this problem can be resolved. Discussion of this and concepts of the general network model is given in the paper by Taylor and Frank [see page 67].

At the more theoretical end of the spectrum is the class of data models based on

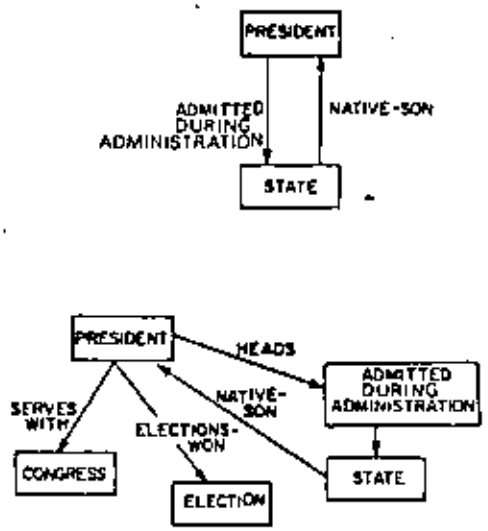


FIGURE 4. Some network structure examples.

mathematics, especially on set theory:

- relational,
- binary association, and
- extended set models.

The *relational* model [M3] deals with entities which have no containment. Thus each entity is made up only of items. The notation introduced earlier can be used to define the same three entities (two of them unchanged):

```
PRESIDENT  = (PRES-NAME, SPOUSE)
BIRTH-DATE = (MONTH, DAY, YEAR)
CHILDREN   = (C-NAME, DATE-OF-BIRTH).
```

But there are now no links between the three entities. These may, however, be made explicit by using the *candidate keys* to establish the relationships:

```
P-DATE-OF-BIRTH = (PRES-NAME, MONTH, DAY, YEAR)
P-KIDS-OF       = (PRES-NAME, C-NAME)
```

assuming that the candidate keys (unique by definition) are PRES-NAME, MONTH, DAY, YEAR, and C-NAME. Another method is to link the entities implicitly by passing the owner candidate key (PRES-NAME) into the dependent entities:

```
PR-BIRTH-DATE = (PRES-NAME, MONTH, DAY, YEAR)
PR-CHILDREN   = (PRES-NAME, C-NAME, DATE-OF-BIRTH).
```

The instances of a group are often called *n-tuples* in the literature of relational systems, which are discussed in the paper by Chamberlin [see page 43]. Relational systems have been in use for some time at universities and research laboratories, e.g., the use of MACAIMS [Z1] and ADMINS [Z2] at MIT, and of RDMS [Z3] at General Motors Research. Some prototype systems are appearing on the market now.

The binary association model, as discussed by Senko [DL5], is part of an attempt at understanding and formalizing data semantics through the use of binary relations.

Although one of the earliest set processors was proposed in the Information Algebra [M1], Childs' set model [M2] was one of the first to be implemented, and it is also being investigated by Hardgrave [M4]. The extended set allows storage of a very wide range of ordered sets and ordinary sets, and is intended to provide maximum generality in storing relationships. However, application of these models is still in the realm of research, though one commercial system is now available [V24].

To recapitulate, information structuring (the selection of entities and specification of relationships between them) is a modeling process with little methodology, other than common sense. In order to use a DBMS, the information structure must be mapped to the logical structure of the system. The mapping is expressed in a DDL. The instances of the data (the data base) are stored by the DBMS to conform to this logical structure. A DBMS generally supports only one of the data models: relational hierarchy, or network. Since each model uses a different terminology, Table 1 attempts to equate the various terms used with the concepts that have been developed in this section.

The criteria for designing and selecting a "best" model has not yet been established—nor is it likely to be established in the near future. The user is therefore faced with two decisions: which data model to utilize (i.e., which type of DBMS), and how to structure the data using the chosen model.

Concept	Relational	Network	Hierarchic
Item	role name/domain	data item type	item/field
Item value	component	data item occurrence	value
Group	not allowed	group	group
Entity (type)	relation	record type	entry/segment type
Entity instance	tuple	record occurrence	entry/segment occurrence
Relationship	foreign key comparable underlying domains	set type	hierarchic (implied)
Relationship instance		set occurrence	assembly
Data administrator view	data model	logical structure	logical structure
Definition of data administrator view	data model definition	schema	schema
User view	data submodel		
Definition of user view	data submodel definition	subschema	subschema
Data-base subdivision		area	
Entry points	primary key	singular sets CALC records	root group root segment
Single unique/identifier	candidate key	key	sequencer (unique)

TABLE 1. COMPARATIVE TECHNOLOGY.

Mapping from the Logical to the Physical Structure

The need to create and load a data base, i.e., to make the data definition and then populate it with data, leads to the physical structure, which is the representation of data in storage. The accessing process for the data base management system is shown in somewhat oversimplified form in Figure 5. The definition of the logical structure is stored within the DBMS and associated with the request so that any logical relations may be derived. As an example, the request:

PRINT SPOUSE WHERE PRES-NAME = "FORD"

does not mention that we are dealing with a PRESIDENT entity; it is left to the DBMS to discover this fact from the logical structure. The physical mapping must have some mechanism that will determine which data to retrieve (using the key PRES-NAME if possible), and then will call the relevant operating system access method and apply any deblocking that is necessary to return the required portion of the character stream.

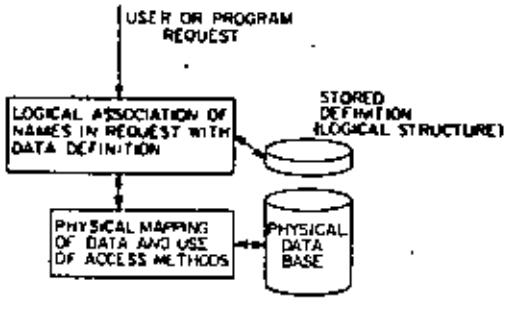


FIGURE 5. Logical and physical aspects of a DBMS.

The process of mapping from occurrences of data to their bit-string representation on disk or tape is generally system-dependent; therefore, these factors are discussed in the separate papers in this issue of SURVEYS. Most DBMS format (block and manage) the pages or records themselves, and most use the operating system access method to store and retrieve the data from secondary devices.

In fact, because most modern DBMS use the available operating system, they gen-

erally use many of its facilities. Therefore, communication management facilities, program library management, access methods, job scheduling, special program management (e.g., sorting and compiling), concurrent access prevention, checkpoint facility, etc. typically are all "adopted" by the DBMS, though some rewrite and additions may be necessary.

4. HISTORICAL PERSPECTIVES

The origin of DBMS can be traced to the data definition developments, the report generator packages, and the command-and-control systems of the fifties—a time when computers were first being used for business data processing. Many systems have been developed since the fifties (See the surveys by Minker, [U1, 4]). MITRE [U3, 8] and CODASYL [U2, 7] show numerous system implementations that have generated wide interest among users.

In 1969 Fry and Gosden [U5] analyzed several DBMS and developed a three-category taxonomy: Own Language, Forms Controled, and Procedural Language Embedded. Succinctly stated, these categories can be contrasted as follows: *Own Language Systems* (such as GIS [V10]) have a high-level, unconventional programming language; *Forms Controled Systems* (such as MARK IV [V12]) use the "fill-in-the-blank" approach, and *Procedural Language Systems* (such as I-D-S [V9]) take advantage of existing higher-level programming languages.

In 1971 the CODASYL Systems Committee [I6] observed that the most significant difference among DBMSs was the method employed in providing capabilities to the user. The Committee developed a two-category classification scheme, *Self Contained* (which included the Forms Controled category) and *Host Language*.

It is impossible to survey all systems, but it is possible to trace the evolution of the DBMS by tracing the evolution of two precursors of data base management: data definition languages and the development of generalized RPG systems.

Evolution of Data Definition Languages

One important factor in the evolution of DBMS is the development of data definition languages. They provide a facility for describing data bases that are accessed by multiple users and by diverse application programs.

Centralized Data Definition: Fifties and Sixties

Probably the first data definition facility was the COMPOOL [DL1] developed at the MIT Lincoln Laboratory for the SAGE Air Defense System in the early fifties. COMPOOL provided a mechanism for defining attributes of the SAGE data base for its hundreds of real-time programs. The COMPOOL concept was later carried over to JOVIAL [PL4] (a programming language), but some of the capability was lost when the language was implemented under a generalized operating system; the data definition became local to the language rather than global to the system.

About the same time, hardware vendors were developing programming languages for business applications: FACT [PL1] was developed by Honeywell, GECOM [PL3] by the General Electric Company, and Commercial Translator [PL2] by IBM; all provided some form of data-definition facility. GECOM and Commercial Translator provided the capability of defining intrarecord structures, and FACT offered the more advanced capability of providing inter-record hierarchical structures.

Under the aegis of CODASYL, these vendor efforts were merged into COBOL [PL5] in the late fifties. This language has a centralized DATA DIVISION which achieves the separation of the description of data from the procedures operating on it. While the DATA DIVISION initially mirrored the data as stored on tape or cards, implementors soon found themselves using different ways of physically storing data. This inherent incompatibility between physical data stored by different manufacturers becomes an important factor when data must be exchanged between two systems.

Approaches which attempt to mitigate the

data-transfer problem are the subject of recent research on the description of physical structures and the development of stored-data definition languages.

Stored-Data Definition Languages: the Seventies

One of the first efforts in this area was mounted by the CODASYL Stored-Data Definition and Translation Task Group [SL2] in 1969 with the goal of developing a language to describe stored data. At the 1970 ACM SIGFIDET (now SIGMOD) meeting, a preliminary report was made [SL3], and later reports were published in 1972 [SL5]. Notable basic research efforts in the development of these languages were reported by Smith [SL1] and Sibley and Taylor [SL4, 7] in 1971.

The Data Independent Accessing Model (DIAM) [DL3], developed by Senko and his colleagues at the IBM San Jose Research Laboratory, provides a multilevel data description capability. The description starts at the information level, structures this into a logical definition, adds encoding information, and ends with a physical description of the storage device and its logical-to-physical mapping structure. Each level provides augmentation of the description at the preceding level. Recent work by Senko [DL4,5] extends the information level in a new language called FORAL.

Thus, the single-level data description facility of the fifties, made incompatible by storage developments in the sixties, led to the recent development of stored-data description facilities in the seventies.

Development of Report Generator Systems

The development of programming languages originally allowed the user (a programmer) to define reports by giving simple definitions of the format of the lines and then writing procedures to move data into buffers prior to printing each line. Therefore, the program written to produce a complete report could consist of large numbers of statements involving expensive programming. The development of *report generators* stems from a need to produce good reports without this large programming effort. In most cases, report generators can perform complex table transformations and produce sophisticated reports from a data base. These, then, allowed the user to examine and manipulate large volumes of data, and they may be said to be a precursor, or a particular type of modern DBMS.

The Hanford/RPG Family (Figure 6)

The patriarch of today's RPG system was developed at the Hanford (Washington) operations of the Atomic Energy Commission, which was then managed by the General Electric Company. In 1956 Poland, Thompson, and Wright developed a generalized report generator [G1] (MARK I) and a generalized sort routine for the IBM 702. The capability was extended in 1957 by the development of a report and file maintenance generator (MARK II). These routines provided the basis for a joint development by several users under the SHARE organization of the 709 Package (9PAC) [W1] for the IBM 709/90.

9PAC is the principal ancestor of most commercial report generators developed since 1960. Foremost among these is the Report Program Generator (RPG) developed for the IBM 1401 in 1961; this has evolved into the RPG for the IBM System/360 and an enhanced RPG II for the IBM System/3, System/360, and several other computers [W2, 3]. Other members of the Hanford family include the COGENT systems, developed by Computer Sciences Corporation for the IBM 709 and System/360 between 1965 and 1969 [Y5], and the SERIES system [Y9].

Another system, also based on MARK II ideas, was being defined during the late fifties in a SHARE 704 Project under Fletcher Jones. This IBM 704 system, called SURGE [W4], was the predecessor of GIRLS, the partiarch of the Postley/MARK IV family.

5. DEVELOPMENT OF DBMS

The development of the data-base management systems may be divided into three somewhat overlapping periods: the early developments, prior to 1964; the establish-

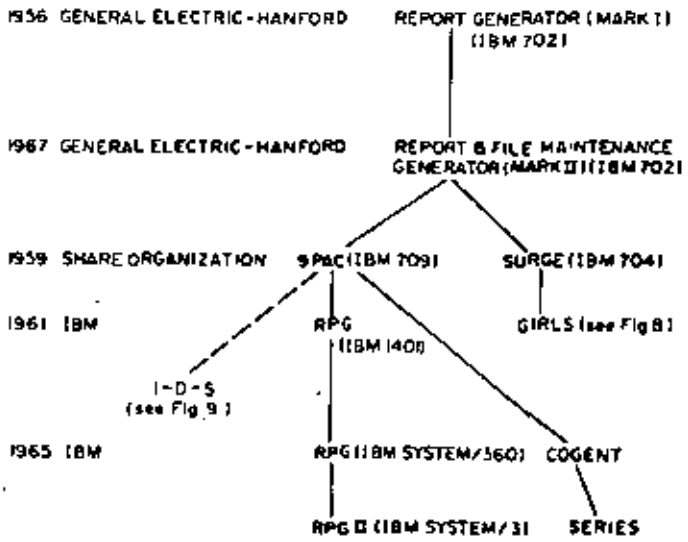


Figure 6. The Hanford/RPG Family.

ment of families during the period 1964–1968; and the vendor/CODASYL developments from 1968 to the present. Since the characteristics of the data-base management systems differ considerably during these periods, we discuss them separately.

Early Developments: Prior to 1964

The impetus for DBMS development came originally from users in government, particularly from the military and intelligence areas, rather than from industry and computer manufacturers. Although these prototypes bear little resemblance to today's systems and were somewhat isolated, they provided some interesting "firsts" in the evolution of data-base technology. They also provided the beginnings of several significant DBMS families.

In 1961 Green [X2] and his colleagues developed a natural-language system called BASE-BALL. Though not a data-base management system by current definition, it made a contribution to the technology by providing access to data through a subset of natural language (a limited vocabulary of baseball-related terms). At approximately the same time, the first implementation of a B-Tree was described by Cellilla and Sams [X6].

Cheatham and Warshall were probably the first to discuss the translation of a query language. They designed a language, QUERY [X7], and developed techniques for analyzing its syntax and compiling statements into machine code.

One of the first identifiable data-base management systems to appear in the literature was an elegant generalized tape system developed by Climenson for the RCA 501 in 1962. This system, called Retrieval Command-Oriented Language [X8], provided five basic commands, with Boolean statements permitted within some of them. The user had to specify the data description with the query so that a program could be bound to its data.

Another early and ambitious development was ACSI-MATIC [X1] sponsored by the US Army in the late fifties. This system was designed by Minker to emphasize effective memory utilization and inferential processing. It could make inferences such as: if John is the son of Adam, and Mary is the sister of John, then Mary is the daughter of Adam. It contributed the first generalized data-retrieval accessing package for a disk-oriented system with batched requests, a dynamic storage algorithm for managing core storage, and the first assembler to use a dynamic storage allocation routine. Because disks were not reliable at that time, the ASCI-MATIC system was never fully imple-

mented. A prototype version was implemented later at RCA (1964).

The US Air Force also pioneered development of DBMS by sponsoring several projects at the MITRE Corporation. The prototype, called Experimental Transport Facility (ETF), led to the Advanced Data Management System (ADAM) [X24, 25, 27, 33], initiated in 1962. ADAM was implemented on an IBM 7030 (STRETCH) computer, with the design goals of providing a laboratory for modeling, prototype development, design verification, and evaluation of DBMS. Although ADAM did not meet all its ambitious design goals [X37] (many have not yet been achieved anywhere), it still remains a significant contribution to the technology.

The COLINGO system [X22], a contemporary of ADAM, was really a series of tape-oriented data-base systems with COBOL-like logical data structures implemented on the IBM 1401 computer system. The C-10 [X30] system, originally named as a follow-on to COLINGO, was implemented on the IBM 1410 computer and embodies many of the ADAM concepts. The influence of ADAM can be seen in System Development Corporation's LUCID [X13, 21], and in parts of Auerbach's DATA MANAGER-1 (DM-1) [X31].

Establishment of Families: 1964–1968

During this period the isolated developments diminished and full-scale families of DBMS emerged, some borrowing heavily from the past, others from sibling developments. A family is not limited to one company or government agency; because of the mobility of its developers, a family may spread across organizations, providing cross-fertilization of ideas. Although the family lineages of DBMS are sometimes intertwined, each can be traced to its progenitor.

The Postley/Mark IV Family (Figure 8)

One early system, which evolved into the MARK IV family, was GIRLS (Generalized Information Retrieval and Listing System), developed for the 7090 by Postley [X4]. Influenced by the SHARE SURGE development (as discussed on page 20), Gesc led successively to the development of the MARK I, MARK II, and MARK III systems for the IBM 1401/60 at Informatics between 1961 and 1967. In 1968 the highly successful MARK IV SYSTEM [V12] was released for use on the IBM System/360. Since then, numerous releases of MARK IV have provided over twenty new features, and MARK IV has now been implemented on other hardware.

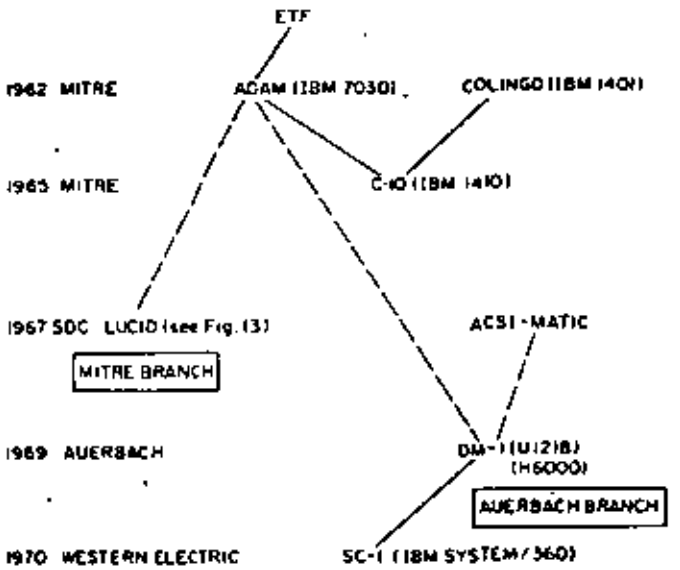


FIGURE 7. The MITRE/Auerbach Family.

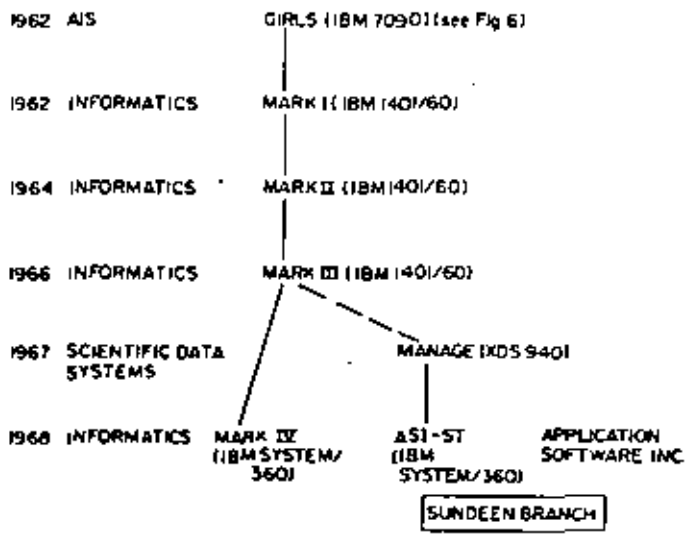


FIGURE 8. The Postley/MARK IV Family.

A significant offshoot of the Postley/MARK IV family is the Sundeen branch. This spans two different companies, starting with the MANAGE System [X23, Y4] developed at Scientific Data Systems, and followed by the AS-IST system [V1] developed at Applications Software in 1967 for the IBM System/360.

Bachman/IDS Family (Figure 9)

The Integrated Data Store (I-D-S) [X15, 18] was developed by Bachman and his colleagues at the General Electric Company in 1964. The I-D-S system, which stems from the same needs as 9 PAC, combined random-access storage technology with high-level procedural languages (GECOM in 1963, and COBOL in 1966) to provide a powerful network model of data. Significant I-D-S developments included:

- new data manipulation verbs or procedure calls at the high-level language interface;
- separate storage- and program-level item descriptions;

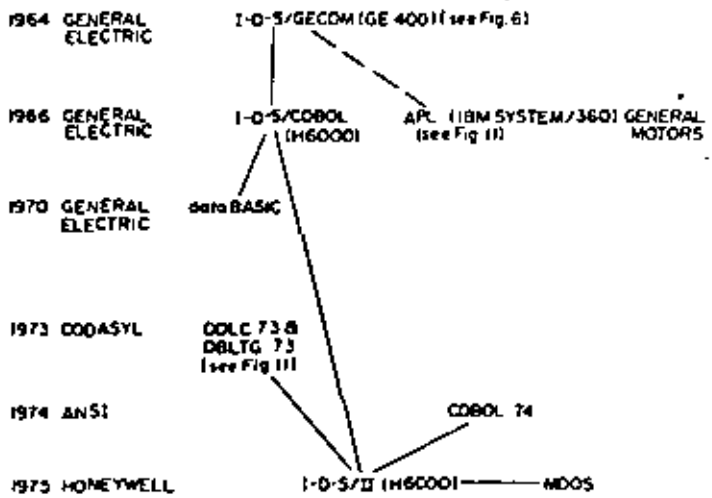


FIGURE 9. The Bachman/IDS Family.

- implicit insertion and removal of groups from relationships, based upon selection and ordering rules;
- retrieval of, and modification to both primary and secondary keys;
- data paging concepts based on logical data-base keys;
- incremental recovery and restart using "before" and "after" images; and
- shared access to the data base, with automatic detection of interference and automatic restart capability.

Since 1964 the I-D-S system has evolved under several different hardware systems, operating systems, and host languages. Recently, a new version, I-D-S/II [V9], using COBOL 74 [P1,7], has been made available by Honeywell. It is consistent with the CODASYL DDLC 73 specification [S3] which will be discussed in the section on the CODASYL/DDLC 73 specification [S3] which will be discussed in the section on the CODASYL/DBTG Family, on page 25, and with recent COBOL additions.

In 1966 Dodd and his colleagues at General Motors Research developed APL (Associative PL/I) [X28], which is a development somewhat similar to that of I-D-S, but intended to provide data-management functions for a computer-aided design environment [G11]. APL provides six data-manipulation verbs in a PL/I host-language environment: CREATE, INSERT, FIND, FOR EACH, REMOVE, and DELETE. Another contribution of APL was the introduction of a distinct technology which separated logical relationships of the owner and member groups from their physical implementation.

Another branch in the I-D-S family is the dataBASIC system [V11], implemented by Dressen at General Electric (now Honeywell) in 1970. This system offered the nonprogramming user high-level access to homogenous files (single record type) in a time-sharing environment using the BASIC programming language. Its only retrieval statement consists of the FOR (Boolean search statement), which qualifies a set of groups (records) to be retrieved. Each retrieval is processed by any number of processing statements until a concluding NEXT statement is encountered.

A recent offshoot in the I-D-S family is the Honeywell Management Data Query System, MDQS [V10]. This system is a self-contained query and report specification facility to access sequential, index sequential, and I-D-S files.

Formatted File/GIS Family (Figure 10)

At about the same time as the host language progenitor (9PAC) was evolving, a series of government systems was being developed to

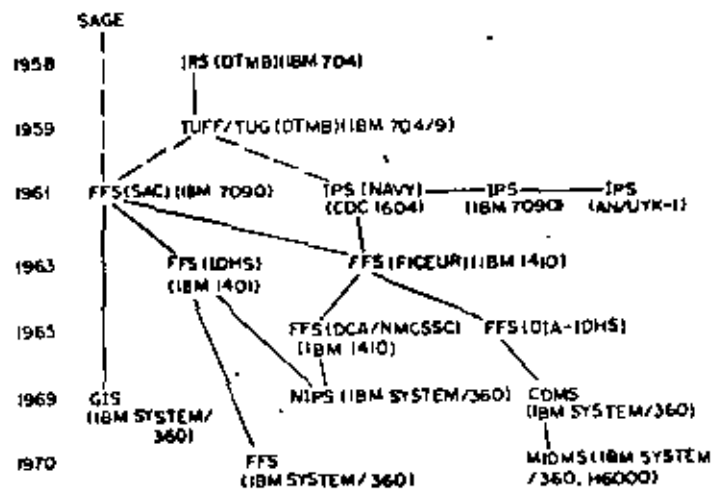


FIGURE 10. The Formatted File/GIS Family.

support the needs of the Command-and-Control and the Intelligence communities. Perhaps the most prolific of these was the Formatted File family, which spans all three development periods. Its origins can be traced to a series of systems developed at the David Taylor Model Basin[4] by Davis, Todd, and Vesper. One of the principal systems—Information Retrieval (IR) [X3, 9]—was an experimental prototype developed in 1958 for the IBM 704. This was followed by two formatted file-processing packages: Tape Update for Formatted Files, TUFF [X16, 20], and Tape Updater and Generator, TUG [X5] (both developed to run on the IBM 704). Later this family split into two branches in the Air Force and Navy. The Air Force branch, SAC/AIDS Formatted File System [X14], was developed in 1961 for the Strategic Air Command 438L system. Its major contribution to data technology was the development of a file format table, i.e., a "self describing" data base. By storing a machine-readable data definition with the data, each data base was directly accessible by FFS.

The Navy branch, Information Processing System (IPS) [X11, 12, and Y10], was also developed in 1963 for the CDC 1604 by NAVCOSSACT. IPS also made contributions to data-base technology in the implementation of a multilevel hierarchically structured data base on sequential media, and in its implementation on several different hardware systems, such as the IBM 709/90 [X19] and the AN/FYK-1 [X32].

During the implementation of IPS in 1963, another branch of the family was developed for the Naval Fleet Intelligence Center in Europe (FICEUR) [X10]. This FFS was patterned after the SAC FFS and implemented on the IBM 1410. SAC also added an FFS on the IBM 1401 for the Pacific Air Force Headquarters. This system was later reprogrammed for the IBM System/360 and is still in use on smaller models.

About 1965 the SAC and FICEUR branches of the formatted-file family merged, resulting in the NMCS Information Processing System (NIPS) [X17]. NIPS added the concepts of logical file maintenance, improved query language, and on-line processing. In 1968 NIPS was converted from IBM 1410 to IBM System/360 and named NIPS-360 [Y12].

A cousin of NIPS was also developed for the intelligence community—the Intelligence Data-Handling Formatted File System [X26]. This emphasized efficient large-file processing and provoked interest in machine-independent implementation using COBOL. Prototype development of such a system began in 1968 by the Defense Intelligence Agency. The effort was first named the COBOL Data Management System (CDMS) [Y8]; later (1970) it was renamed the Machine Independent Data Management System (MIDMS) [Y11]. It was originally implemented on the IBM System/360 and was later coded (in 1973) for the H6000 series.

SAC FFS is considered to have inspired IBM's Generalized Information System (GIS) [Y16, 17]. This was originally developed as a stand-alone program product for System/360 (1965), but has been extended and enhanced to act as either a stand-alone system or ad hoc interrogation interface for the IMS family.

Vendor/CODASYL Developments: 1968 to the Present

The trend in this period shifts from in-house family-oriented activities to proprietary vendor development. As a result, some advances made by commercially available DBMSs disappeared into a veil of secrecy. While few references have appeared recently on the internals of particular DBMSs, the technical literature abounds with articles on mathematical and theoretical aspects, especially of relational systems. Chamberlin's article (see page 43) provides an excellent bibliography of this development. Recent years also show the entry of CODASYL into the data-base field.

CODASYL/DBTG Family (Figure 11)

Based upon the pioneering ideas of I-D-S and APL, the CODASYL Programming

[4] The David Taylor Model Basin is now called the David Taylor Naval Ship Research and Development Center.

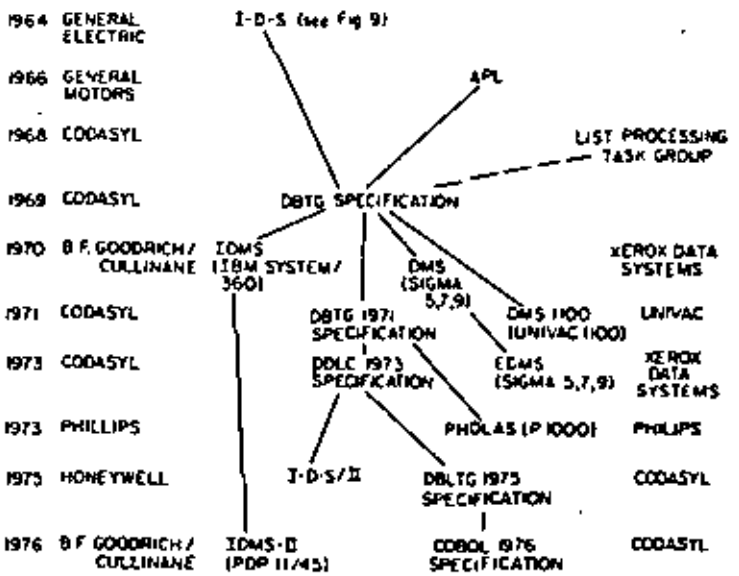


FIGURE 11. The CODASYL Family.

Language Committee started a new task group to work on a proposal for extending COBOL to handle data bases [PL6]. This group was originally called the List Processing Task Group, though its name was later changed to the Data Base Task Group—DBTG—its major acronym, which will be used here. The first semipublic recommendations of the DBTG were made in 1969 [S1]. These recommendations detailed the syntax and semantics of a Data Description Language (DDL) for describing network-structured data bases, and the definition of Data Manipulation Language (DML) statements to augment COBOL. The task group intended that the DDL specifications should be available to all programming languages, while extensions like the DML would be needed for every language.

The initial DBTG specification was reviewed by many user and implementation groups. Their recommendations were further considered, and a new report was issued in 1971 [S2]. The major change involved separation of the data description into two parts; a Schema DDL for defining the total data base, and a Sub-schema facility for defining various views of the data base consistent with different programming languages.

Based on the reviews of the 1971 report, CODASYL took two significant actions:

- a new standing committee was created to deal exclusively with the data description, the Data Description Language Committee (DDLC); and
- the DBTG was replaced by a new task group to deal only with COBOL extensions, the Data Base Language Task Group (DBLTG).

Since that time, a new subcommittee has also been formed to add DML statements to FORTRAN.

The DDLC was charged with taking the Schema DDL and developing a common data description language to serve the major programming languages. In January 1974 a first issue of the Data Description Language Committee's publication, the Journal of Development, was published [S3]. This report specifies only the syntax and semantics of the DDL.

The DBLTG was charged with making the 1971 report of the DBTG consistent with CODASYL COBOL specifications. In February 1973 the DBLTG submitted its report to the CODASYL Programming Language Committee. This report is very similar to the 1971 DBTG report, with nomenclature and relatively cosmetic changes. New items in the 1973 report included an ex-

tension to the facility for dealing with error returns.

Implementation of systems which conformed to the 1969, 1971, and 1973 DBTG specifications started in 1970 with the UNIVAC DMS 1100 [V22] for the 1108, and since then for the UNIVAC 1110 series computers. At about the same time, B. F. Goodrich implemented a system called Integrated Data Management System, IDMS [V7], for the IBM System/360. This has since been extended to IDMS-11 for the Digital Equipment Corporation PDP 11/45. The IDMS series is marketed by Cullinane Corporation. The Digital Equipment Corporation has implemented DBMS-10 [V8] for its PDP 10 computer system.

Some extensions to self-contained facilities for ad hoc interrogations have been implemented by Control Data Corporation, Query/Update [V6], and by Xerox Data Systems, EDMS [V23]. In the Netherlands, Philips implemented a family of systems termed PHOLAS [V19], and in Norway the SIBAS [V20] system has been developed by Shipping Research Services. Honeywell has updated I-D-S to conform to 1973 specifications; this is the I-D-S/II [V9].

IMS Family (Figure 12)

The IMS family of systems is an outgrowth of the Apollo moon-landing program. Its origins can be traced to two developments at The Space Division of North American Aviation (now Rockwell International) in 1965. One was the implementation of GUAM, (Generalized Update Access Method), the forerunner of Data Language/One (DL/I). The other was the implementation of two teleprocessing applications, EDICT (Engineering Document Information Collection Task) and LIMS (Logistics Inventory Management System). The software package which supported EDICT and LIMS, the Remote-Access Terminal System (RATS), was jointly developed by Rockwell International and IBM during 1964–65. Both GUAM and RATS were originally implemented on the IBM 7010 with 1301 disk storage.

In 1966, IBM, Caterpillar Tractor Corporation, and Rockwell International agreed to a joint development effort to produce a DBMS, the Information Management System (IMS) for the IBM System/360. When the system had to be frozen in 1968 (to meet the Apollo commitment), Rockwell and IBM each continued with separate developments, while Caterpillar withdrew entirely from the effort. The development at Rockwell took the name of Information Control System/Data Language/I (ICS/DL/I).

Originally, DL/I [X35] was a data description facility which provided a means for describing and organizing a hierarchically structured data base. It also provided interfaces, which the programming user invoked to access and store data from the host language (originally COBOL). The on-line component, ICS/DL/I [X34], added in 1968, allowed multiple access by using the DL/I interface from COBOL or PL/I programs. In addition to running teleprocessing simul-

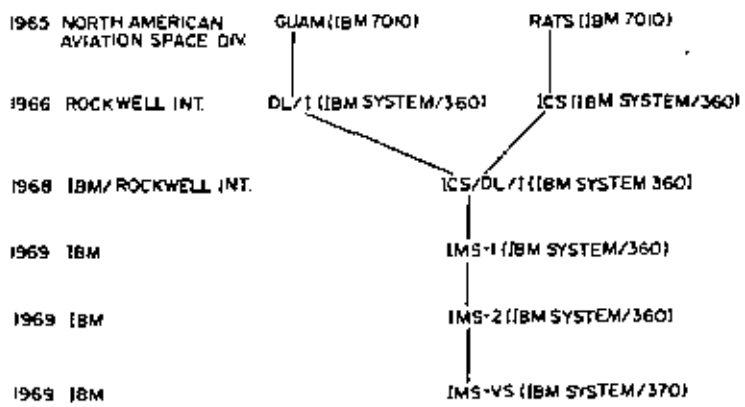


FIGURE 12. The IMS Family.

taneously with batch processing, the system handled several remote terminals.

In 1969 IBM released its version, the Information Management System/360 (IMS/360) [V13]. Since 1969 a series of improvements has marked its evolution [V14, 15].

Inverted File Family (Figure 13)

Following the Lucid System development in 1968, the Advanced Research Project Agency (ARPA) sponsored System Development Corporation's development of the Time-Shared Data Management System, TDMS [Y1, 2, 3, and X29]. This was designed to operate in the time-sharing environment of the Adept executive on the IBM System/360. It was the first DBMS to combine an inverted file implementation of hierarchical data model with interactive processing.

In 1966 the Computation Center at the University of Texas began the development of a Remote File Management System (RFMS) [Y7] on its CDC 6000. RFMS differed from TDMS mainly in its internal design. A version of RFMS was marketed by CDC as Mars VI [V5].

MRI Systems Corporation (whose principals were originally associated with the University of Texas) continued development of an RFMS under the name of System 2000 [V18], which was offered commercially in 1970. A number of significant enhancements have been made since 1970 so that System 2000 offers an integrated set of host-language and self-contained capabilities.

Additional Vendor Developments

A variety of other data-base management systems based on inverted files for efficient query processing were developed during this period by other vendors. Two of the more commonly known are Adabas [V21], developed by Schoell at Software AG (West Germany), and Model 204, developed by Computer Corporation of America [V4].

Adabas uses the inversion tables not only for efficient retrieval, but also for linkages between records of different files. Adabas provides access to the data through a host language interface, a self-contained language for on-line inquiry, and a batch report generator. Adabas is one of the few systems which offer a data compression facility.

The Model 204 query language provides most of the power of a general-purpose programming language from an on-line terminal, but is easy to use for simple requests. This system uses the Isam access method to allow multiple field indexing and variable length records for file compression as well as for text processing.

Three other vendor developments date back to about 1969, Total, DM-1, and DMS II. Although in its initial release, Total [V3] was primarily a direct access data-base management system, facilities were soon added to process DBTG-like sets implemented with chain pointers. Total is a host-language system, which can model the major data structures of the DBTG specifications, and it was one of the first systems to offer a Schema-Sub-schema processor fa-

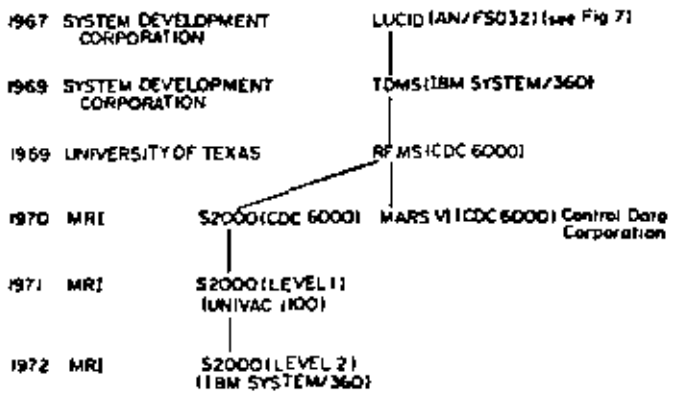


Figure 13. The Inverted File Family.

cility. It has become one of the most widely used data-base management packages today.

The Data Manager-1 System (DM-1) [X31], designed by Sable at the Auerbach Corporation, stems from the Army ACSI-MATIC development and MITRE'S ADAM. DM-1 consists of a series of service routines for returning and storing data; using these routines, both high-level ad hoc user functions and host-language application programs can be developed. DM-1 was implemented at the Air Force Rome Air Development Center on U1218 computer and the Honeywell H6000. Based on the design philosophy of DM-1, the Western Electric Company, initially assisted by Auerbach, developed System Control-1 [Y6] on the System/360.

Another development, by the Burroughs Corporation, is the Data Management System II [V2] for the B6700/B7700 computer. Basically a host-language type system using COBOL, its data definition language is formed in set-theoretic terms. It also offers a storage definition option.

6. THE PRESIDENTIAL DATA BASE EXAMPLE

The discussion of data-base models in other articles in this issue of COMPUTING SURVEYS will use a unified example which deals with some parts of the Executive branch of the US Government, with data about the President, his Administration, elections, Congress, etc. We use this example because it is almost self-explanatory; it was first enunciated in a paper by Willner, et al. [G9].

Because the example deals with the Executive branch, the most obvious entity is the PRESIDENT. The important items in the PRESIDENT entity will be assumed to be: the President's name (PRES-NAME), BIRTH-DATE and DEATH-DATE, the party affiliation (PRES-PARTY), and the name of his SPOUSE. It will also be considered necessary to know the STATE-NAME of which the President is a native son. However, since STATE will later be defined as an entity, we could alternatively define a relationship NATIVE-SON between PRESIDENT and STATE.

Using the notation presented in Section 3 under the discussion of the "Elements of Logical Structure" (page 13) we have Display 1 below. If, however, an explicit relationship were to be used for the native son, and STATE-NAME is the key of STATE then the statement appears as in Display 2 below.

The next entity of interest is the President's ADMINISTRATION, which contains items such as the administration number (ADMIN-NUMBER) (e.g., George Washington was No. 1), the inauguration date (INAUG-DATE), and the Vice-President (VP). In order to identify the President of each Administration, it is also necessary to include the item PRES-NAME in the ADMINISTRATION entity.

At this point, it is worth asking why the PRESIDENT entity does not *contain* the ADMINISTRATION entity. This is a design decision, and the reader must assume it is based on consideration of usage and modeling. It should be noted, however, that a President can have had more than one Administration, and consequently, if ADMINISTRATION is contained, it would need to be a *repeating group*. As another alternative, we could assume that the two separate entities have a relationship HEADED between ADMINISTRATION and PRESIDENT. Thus, we have Display 3) below.

Display 1:
PRESIDENT = (PRES-NAME, BIRTH-DATE, DEATH-DATE, PRES-PARTY, SPOUSE, STATE-NAME)

Display 2
PRESIDENT-1 = (PRES-NAME, BIRTH-DATE, DEATH-DATE, PRES-PARTY, SPOUSE)
and
NATIVE-SON = (PRES-NAME, STATE-NAME).

Display 3
either
(ADMINISTRATION) = (ADMIN-NUMBER, PRES-NAME, INAUG-DATE, VP);
or:
PRESIDENT-2 = (PRES-NAME, BIRTH-DATE, DEATH-DATE, PRES-PARTY, SPOUSE, STATE-NAME, [(ADMIN-NUMBER, INAUG-DATE, VP)]);
or:
ADMINISTRATION-1 = (ADMIN-NUMBER, INAUG-DATE, VP)
HEADED = (PRES-NAME, ADMIN-NUMBER).

The next entity is that of the ELECTION. The interesting items in the election are: the year (ELECTION-YEAR), the presidential votes in the Electoral College (PRES-VOTES), the LOSER, the LOSER-PARTY, the year in which the party was first created as a political entity (PARTY-FIRST-YEAR), and the votes of the losing party (LOSER-VOTES). Once again, because elections are won by a President, the election entity may have to contain the PRES-NAME; otherwise there must be some relationship WON between the PRESIDENT and the ELECTION entities. Thus, the alternatives are:

ELECTION = (ELECTION-YEAR, PRES-NAME, PRES-VOTES, LOSER, LOSER-PARTY, PARTY-FIRST-YEAR, LOSER-VOTES), etc.

Another entity within the data base is the STATE. It has a name (STATE-NAME), a population (POP), and a number of votes in the Electoral College (STATE-VOTES). States are admitted to the Union during some Administration. This fact may be shown either implicitly, by having some relationship (ADMITTED-DURING) between the ADMINISTRATION and STATE entities, or explicitly, by including the ADMIN-NUMBER in the STATE entity. It might be noted that there is already a link between the PRESIDENT and STATE entities because the NATIVE-SON relation has been shown as an element (STATE-NAME) in the PRESIDENT entity.

We have now defined most of the data base, and need only incorporate the entity CONGRESS to complete it. This entry will contain items such as: CONGRESS-NUMBER, SENATE-REPUBLICAN-PERCENT, SENATE-DEMOCRAT-PERCENT, HOUSE-REPUBLICAN-PERCENT, AND HOUSE-DEMOCRAT-PERCENT. Again, there is a relation between the PRESIDENT and CONGRESS, which may be found explicitly by incorporating PRES-NAME in the CONGRESS entity, or implicitly by arranging a relation CONGRESS-SERVED between the entities.

Figure 14 shows a sample of the presidential data base in tabular form. Unavailable information is shown by a ϕ, e.g., in the Death and Inauguration Date columns.

But there are some drawbacks to this example: one is the fact that it represents a relatively constant data base, for although a President may be replaced, the data about the Administration is still retained. Consequently there is little *updating* in our example, though there may be substantial *addition* to the data base in election years. Some business data bases, however, present a greater propensity to change. For example, a payroll data base regularly has changes to many items such as YEAR-TO-DATE-PAY (presumably after every payday) and SALARY (presumably after every increase). Thus, the presidential data base, while forming the major example, will not suffice alone. Other authors contributing to this issue of COMPUTING SURVEYS will introduce other examples to illustrate particular fine points.

7. TRENDS AND ISSUES

Historically, we have traced the development of DBMS from the early systems, which supported primarily the nonprogramming user for ad hoc requests, to the recent predominance of host-language systems which support the programming user. A current trend is, then, the establishment of a balance—a comprehensive set of DBMS functions for a full spectrum of users while maintaining the current DBMS objectives [F1, 2, and 3]. Some of the current research is developing bridges between various models of data so that a single DBMS can support a variety of data models.

Three major trends and one important issue will affect the future of DBMS: the emergence of conversational systems, the need for geographic distribution of the information system, the technological impacts on DBMS architecture, and the question of standardization of the DBMS interface. Each of these is now briefly discussed.

Ad Hoc versus Programming Systems

Artificial intelligence research has already improved our understanding of the difficulties involved in providing a natural language interface for computers. And though there

has been little that is immediately applicable, the fall-out from this research includes a better understanding of the structure and use of higher-level and very-high-level (restricted natural) language interfaces. As a result, some DBMS already provide good languages for the nonprogrammer who is willing to learn a few rules, and there is growing interest in the development of the casual-user interface (e.g., see [F4]).

PRESIDENT

PRES-NAME	BIRTH-DATE	DEATH-DATE	PRES-PARTY	SPOUSE	STATE-NAME
Eisenhower	10/14/1890	03/28/1969	Republican	Mamie	Texas
Kennedy	05/29/1917	11/22/1963	Democrat	Jacqueline	Mass.
Johnson	08/27/1908	01/22/1973	Democrat	Claudia	Texas
Nixon	01/09/1913	*	Republican	Patricia	Calif.
Ford	07/14/1913	*	Republican	Elizabeth	Mich.

ELECTION

ELECTION-YEAR	PRES-NAME	PRES-VOTES	LOSER	LOSER-PARTY	PARTY-FIRST YEAR	LOSER-VOTES
1952	Eisenhower	442	Stevenson	Democrat	1824	89
1956	Eisenhower	457	Stevenson	Democrat	1824	73
1960	Kennedy	303	Nixon	Republican	1856	219
1964	Johnson	486	Goldwater	Republican	1856	52
1968	Nixon	301	Humphrey	Democrat	1824	191
			Wallace	3rd Party	1968	46
1972	Nixon	520	McGovern	Democrat	1824	17

CONGRESS

CONGRESS-NUMBER	PRES-NAME	SENATE-REPUBLICAN-PERCENT	SENATE DEMOCRAT-PERCENT	HOUSE-REPUBLICAN-PERCENT	HOUSE-DEMOCRAT-PERCENT
83	Eisenhower	50%	49%	49%	50%
84	Eisenhower	49%	51%	47%	53%
85	Eisenhower	49%	51%	46%	54%
86	Eisenhower	34%	66%	35%	65%
87	Kennedy	36%	64%	40%	60%
88	Kennedy Johnson	33%	67%	41%	59%
89	Johnson	32%	68%	33%	67%
90	Johnson	36%	64%	43%	57%
91	Nixon	43%	57%	44%	56%
92	Nixon	41%	54%	41%	59%
93	Nixon Ford	42%	56%	44%	56%
94	Ford	37%	60%	33%	66%

Figure 14. A sample of the presidential data base.

STATE

STATE-NAME	ADMIN-NUMBER	POP	STATE-VOTES
Texas	16	11196730	26
Mass.	ϕ	5689170	14
Calif.	18	19953134	45
Mich.	12	8875083	19

ADMINISTRATION

ADMIN-NUMBER	PRES-NAME	INAUG-DATE	VP
1	Washington	04/30/1789	Adams
2	Washington	03/04/1793	Adams
16	Polk	03/04/1845	Dallas
18	Fillmore	07/10/1850	ϕ
49	Eisenhower	01/20/1953	Nixon
50	Eisenhower	01/20/1957	Nixon
51	Kennedy	01/20/1961	Johnson
52	Johnson	11/22/1963	ϕ
53	Johnson	01/20/1965	Humphrey
54	Nixon	01/20/1969	Agnew
55	Nixon	01/20/1973	Agnew Ford
56	Ford	08/09/74	Rockefeller

Fig. 14. (contd.): A sample of the presidential data base.

A *casual user* is one who uses the system so seldom that all rules and techniques are likely to be forgotten between sessions, hence the need for special treatment. At the other end of the user spectrum are the adept computer programmers who have technical skills and a good knowledge of "system internals." In writing programs for nonprogrammers they presumably utilize all their skills to produce procedures that will run efficiently. The assumption is that programmers cost more (they must be paid while they understand the problem, write code, etc.), but their resulting programs are cheaper to run.

Thus, the case for ad hoc and host-language systems can be considered one of tradeoffs. The following is a partial list of the advantages and disadvantages of the use of higher-level interfaces:

1) Their use facilitates more rapid running of the problem—the user asks the question directly, and he has no need to call on a programmer as intermediary (a process that sometimes takes weeks for even a simple problem in a busy industrial environment). This advantage is offset by the relatively high cost of using what is essentially an interpretive system: the tradeoff is therefore between people and machine costs. The people costs are in programming and debugging, while the machine costs are in running. One presumes that the code produced from a high-level (query) interface costs more to run, therefore the question arises: how many times must the program be run before it pays for the cost of programming? And this is the classical question of compiling, but now in the realm of even higher-level languages and with potentially large data bases. There are, however, very few jobs today which warrant the cost of special (assembler or machine language) programming. This trend continues today in DBMS usage, and the self-contained ad hoc user system is becoming more accepted by the user community.

2) The use of a higher-level language simplifies the structure (removes DO-loops and GO-TO statements) and is generally

more understandable, consequently less error prone. On the other hand, a simple question may invoke a long and costly procedure; e.g., "Give me the average height of all Americans," may involve a sequential search of 200 million records! Also, the possibility for ambiguity immediately arises. The request "Give me the count of all people in New York," could be interpreted as "...all people who are, at this instant, in the state of New York," while the questioner intended to ask "...all people who have,. as their residence, the city of New York." The trouble with this question is obvious, but the user may never realize that the answer given was not correct for the intended question.

3) The very-high-level languages tend to have a mathematical equivalence—they can be transformed into precise mathematical formulas (e.g., in predicate calculus). They are therefore capable of exact checking. In this way, the potentially ambiguous statement can be transformed into an exact statement and "played back" to the questioner, thereby helping to eliminate error. The high-level program, however, does not have an exact statement of its operation in good mathematical terms; it does what the programmer told it to do, good or bad (and all too often the latter). Precision of statement is an advantage to the mathematically sophisticated user, and possibly to others as well.

Thus, the user trend may well be toward the higher-level-language interface, but for years to come it will be necessary to program the large and repetitive systems of industry and government efficiently by using the language interfaces currently in use (e.g., COBOL, FORTRAN, PL/I).

Geographically Distributed Systems

Inexpensive communication between computing systems, and the development of national and international networks have forced further changes on the design of computing systems. In this, DBMS is no exception. The concept of distributed data bases, where a processor calls on data at several other locations, is already a reality on some homogeneous systems—and possibly (with difficulty) on some nonhomogeneous systems. This trend may be seen, in part, as an answer to the wish of industry and government to access its data in reasonable time.

As an example, one major corporation found that the use of a computer network allowed it to strike a corporate dollar balance each Friday; thus, the company could let the money out on short-term loan (over the week-end). Surprisingly, the money realized as interest on the loan paid for all the network facilities. Similarly, in many large corporations, the warehousing cost is great; all material resting in inventory represents an unprofitable capital expenditure. Large retail merchandising companies can reduce inventory costs by knowing *what* is available *where* in their many warehouses, and thus be able to reduce surplus stock. Some large corporations have been able to give their sales forces remote access to their computer systems, thereby allowing the salesman (and through him, the customer) direct on-line access to shipping and pricing information. The competitive advantage is very high in such cases.

Distributed systems, then, show a need for:

1) computers to be networked. It is not generally possible to have all the power at a central site, and each major node (e.g., the largest warehouses) has its processor.
2) data to be distributed. If the data is entered at hundreds of locations throughout the country, it is probably efficient to store it near the entry port. In some large banking systems, the customer accounts are kept in the computer system at the local banks, but other branches can still service the customer (and debit the account!).

But distributed systems pose many new problems, and exacerbate many old ones. Some of the new problems are revealed in the following questions.

- How do we change the request language? Does the user have to know the location of the data? Is there a central data dictionary/directory? Can the user request data by broadcasting a message for it?

- Is it better to store multiple copies? How much extra will it cost to update a data base from a remote location? What parts of the data base should be stored (i.e., how does one distribute the data efficiently)? What are the best places to run a program (it may be cheaper for a user at A to transport data at B to the program at C and then just receive the answers at A)?

The old problems have already been discussed, but are now complicated by the extra complexity of the distributed system:

- What redundancy is necessary to ensure good reliability of both hardware and data? How much does this affect the user in terms of the response time for updates, and the excess processing cost?
- What problems are likely to occur in concurrent operation? The possibility that several users will all contend for the same resources, and consequently will need effective scheduling and control, is obviously more acute in a large, distributed, many-user system.
- How can privacy be retained? The potential for breaking the system rises as its complexity increases. The chance of message interception obviously increases also.

Thus, the trend to distributed data bases, with concepts of data machines as special resource nodes on the network, brings with it a new set of tradeoff decisions.

Data-Base Machines

Distributed data bases, in conjunction with emerging technology, will have a significant impact on DBMS architecture and on the DBMS functions. There already are computers dedicated to DBMS, e.g., the Datacomputer [F5, 6]. "Front-end" and "back-end" computers are in the prototype stage [F7, 8]. Also, new disk technologies and associative devices will have a great impact on DBMS architecture [F9, 10].

To Standardize or Not?

The computing profession has ambivalent feelings about standardization: everyone seems to admit it has merits, but finds excuses in order to stop it from happening too soon in his own field of interest. The arguments for and against standardization (in any area) are now given.

For standards, there is one major argument: The provision of a standard aids the user by making objects interchangeable; the nut, if of the same diameter, fits the bolt. Thus:

- the programming language is the same on all machines: so the programmer who knows COBOL, for example, can be transferred, or may get a new job and not need retraining;
- the company can change machines and run the same COBOL programs, after their recompilation, on the new machine;
- parts are interchangeable: magnetic tapes have standard densities; plug-to-plug compatibility of storage and input/output units is possible;
- data can be interchanged over the network;
- the network protocol is the same, so all users have to learn only one protocol; and
- the commands to enter (log-on) and leave (log-off) the system, and some other controls, are the same throughout the network.

Against standards, there is one major argument: if we do not know the correct technology, standardization may mean costly refitting later, or may even stifle development. This argument is reasonable, since a large-scale data-processing shop may have many thousands of programs representing millions of dollars of investment. Rewriting all these (probably COBOL) programs in some new language is beyond the wishes of most current DP managers, who hope that their programs are "here to stay." Such built-in conservatism will undoubtedly slow down any change from one well-developed standard to another, no matter how good the new standard may be. This stifles acceptance of new ideas.

Many groups are concerned about standardization and are actively working in this area. The DBTG report has been accepted by the Programming Language Committee

of CODASYL as a part of JOD COBOL. The ANSI/X3/SPARC/Study Group on Data Base Systems has been meeting since 1972. Part of their charge is to develop a basis for DBMS standardization. Their recent report [F1] formulates many functional interfaces of a DBMS. The languages used to communicate across these interfaces may be candidates for standardization.

There are therefore many potential areas for standardization of DBMS:

- the definition language for the logical structure;
- the language(s) to manipulate the data;
- the protocols for invoking procedures on the data-base machine;
- the protocols on the network of a distributed system; and
- the storage devices and physical mapping of data.

Each of these has its proponents and opponents for various kinds of system models. As a result, the issue of standardization is a mixture of common sense, politics, economics, philosophy, convenience, and taste. Many researchers consider standards an anathema, but many users see standards as a necessity. The arguments will still be going on fifty years from now (even though there will undoubtedly be DBMS standards by then).

ACKNOWLEDGMENT

The Guest Editor's Introduction to this issue of COMPUTING SURVEYS has already expressed gratitude to the wide variety of experts who made this article possible. This includes developers of DBMS who implemented the systems and who helped us correct errors in the history, as well as our reviewers. V. Kevin Whitney and Richard G. Canning in particular made many valuable suggestions.

CLASSIFICATION OF REFERENCES

A Data Administration
D Data Dictionary
DL Data Definition Language
F Future, Trends
G General
I Introductory
M Data Models—Theory
PL Programming Languages
S DBMS Specifications
SL Stored-Data Definition
T DBMS Texts
U Surveys
V Vendor Systems
W Report Generator
X DBMS Prior to 1968
Y DBMS 1968 to Present
Z Relational Systems

REFERENCES

(A) Data Administration

[A1] EVEREST, G. C., "Data base administrator organizational role and functions," MISRC-WP-73-05.

[A2] GUIDE INTERNATIONAL, "The data base administrator," Nov. 1972.

[A3] CANNING, R. G. (Ed.), "The data administrator function," *EDP Analyzer* 10, 11 (Nov. 1972).

[A4] NOLAN, R., "Computer data bases: The future is now," *Harvard Business Review* (Sept. 1973).

(D) Data Dictionary

[D1] UHROWCZIK, P. P., "Data dictionary/directories," *IBM Systems J.*, 12, 4 (Dec. 1973).

[D2] CANNING, R. G. (Ed.), "The data dictionary/directory function," *EDP Analyzer*, 12, 10 (Nov. 1974).

(DL) Data Definition Language

[DL1] WILWORTH, N. E., "System data control," Tech. Memo. TM222/013/00, System Development Corp., Santa Monica, Calif., August 1975.

[DL2] D'IMPERIO, M., "Data structures and their representation in storage," in *Annual Review in Automatic Programming*, M. Halpern and C. J. Shaw (Eds.), Pergamon Press, Elmsford, New York, 1969, pp. 1–75.

[DL3] SENKO, M.; ALTMAN, E.; ASTRAHAN, M.; AND FEHDER, P., "Data structures and accessing in data-base systems," *IBM Systems J.*, 12, 1 (1973), 30–93.

[DL4] SENKO, M. E., "Data description language in the context of a multilevel structured description: DIAM II with FORAL," IBM Research Report #6769, 1974.

[DL5] SENKO, M. E., "The DDL in the context of a multilevel structured description DIAM II with FORAL," in *Data Base Description*, B. C. M. Douque and G. M. Nijssen (Eds.), North-Holland Publ. Co., Amsterdam, The Netherlands, 1975.

(F) Future, Trends

[F1] WHITNEY, V. KEVIN, "Fourth generation data management systems," *Proc. of AFIPS National Computer Conf.*, 1973, Vol. 42, AFIPS Press, Montvale, N.J. 1973, pp. 239–244.

[F2] BACHMAN, CHARLES, "Trends in data-base Management," *Proc. of AFIPS National Computer Conf.*, 1975, Vol. 44,

AFIPS Press, Montvale, N.J., 1975, pp. 569-576.

[F3] EVEREST, GORDON C., "The futures of data-base management," *Proc. 1974 SIGMOD Conf.*, May 1974, pp. 445-462.

[F4] CODD, E. F., "Seven steps to rendezvous with the casual user," *Proc. IFIP TC-2 Working Conf. on Data Base Management System Compress*, April 1974, North-Holland Publ. Co., Amsterdam, The Netherlands, 1974.

[F5] MARILL, THOMAS; AND STERN, DALE, "The datacomputer—a network data utility," *Proc. of AFIPS National Computer Conf.*, 1975, Vol. 44, AFIPS Press, Montvale, N.J., 1975, pp. 389-395.

[F6] MARILL, T.; AND STERN, D. DATACOMPUTER VERSION I USER MANUAL, Working paper no. 11, Computer Corp. of America, Cambridge, Mass., August 1975.

[F7] CANADAY, R. H.; HARRISON, R. D.; IVIE, E. L.; RYDER, J. L.; AND WEHR, L. A., "A back-end computer for data base management," *Comm. ACM* 12, 10 (Oct. 1974), 575-582.

[F8] HEACOX, H. C.; CONLOY, E. S.; AND COHEN, J. B., "An experiment in dedicated data management," in *Proc. of Internatl. Conf. on Very Large Data Bases*, Sept. 1975, ACM, New York, 1975, pp. 511-513.

[F9] SU, S. Y. W.; COPELAND, G. P.; AND LIPOVSKI, G. J., "Retrieval operations and data representations in a context-addressed disk system," *Proc. ACM-SIGPLAN-SIGIR Interface Meeting on Programming Languages and Information Retrieval*, Nov. 1973, pp. 144-160.

[F10] LIN, C. S., AND SMITH, D. C. P., "The design of a rotating associative array memory for a relational data-base management application," *ACM TODS*, 1, 1 (March 1976), 53-65.

[F11] ANSI/X3/SPARC/STUDY GROUP—DATA BASE SYSTEMS, "Interim report," *ACM/SIGMOD Newsletter:fdt*, 7, 2 (Dec. 1975).

(G) General

[G1] MCGEE, W. C., "Generalization: key to successful electronic data processing," *J. ACM* 6, 1 (Jan. 1959), 1-23.

[G2] MCGEE, R. C.; AND TELLIER, H., "A re-evaluation of generalization," *Datamation*, (July-August 1960), 25-38.

[G3] YAO, S. B.; AND MERTEN, A. G., "Selection of file organization using an analytic model," *Proc. of the Internatl. Conf. on Very Large Data Bases*, Sept. 1975, ACM, New York, 1975, pp. 255-267.

[G4] STEEL, T., "Beginnings of a theory of information handling," *Comm. ACM* 7, 2, (Feb. 1964), 87-103.

[G5] ROSEN, SAUL, "Programming systems and languages—a historical survey," *Proc. of the Spring Jt. Computer Conf.*, 1964, Vol 25, AFIPS Press, Montvale, N.J., 1964, pp. 1-25.

[G6] ROSIN, ROBERT F., "Supervisory and monitor systems," *Computing Surveys* 1, 1 (March 1969), 37-54.

[G7] DENNING, PETER J., "Third generation computer systems," *Computing Surveys* 3, 4 (Dec. 1971), 175-216.

[G8] EVEREST, GORDON C.; AND SIBLEY, EDGAR H., "A critique of the GUIDE-SHARE data-base management system requirements," *Proc. of the 1971 ACM-SIGFIDET Annual Workshop on "Data Description, Access and Control,"* E. F. Codd and A. L. Dean, (Eds.), pp. 93-112, also MISRC-WP-71-2.

[G9] WILLNER, S. E.; BANDURSKI, A. E.; GORDAN, W. C.; AND WALLACE, M. A., "CONRADE data management system," *Proc. AFIPS National Computer Conf.*, 1973, Vol. 42, AFIPS Press, Montvale, N.J., 1973, pp. 339-345.

[G10] BACHMAN, C. W., "The programmer as navigator," *Comm. ACM* 16, 11 (Nov. 1973), 653-658.

[G11] GARTH, W., "Design console technology at General Motors," *Proc. SHARE 1974 Conf.*, August 1974.

[G12] EVEREST, GORDON C., "The objectives of data-base management," *Information Systems COINS IV* (Tou), Plenum Press, New York, 1974, pp. 1-35, also MISRC-WP-71-04.

[G13] NAVATHE, S. B.; AND FRY, J. P., "Restructuring for large data bases: three levels of abstraction," *ACM, TODS*, to appear in June 1976.

[G14] TEOREY, T. J.; AND DAS, K. S., "Application of an analytical to evaluate storage structure", Data Translation Technical Report No. 76DE 7.1. Univ. of Michigan Graduate School of Business Administration, Ann Arbor, 1976.

(I) Introductory

[I1] LYON, J. K., *Introduction to data base design*, Wiley Interscience, Div. of John Wiley and Sons, New York, 1971.

[I2] BACHMAN, C. W., "Data structure diagrams," *SIGBDP: Data Base* 1, 2 (1969).

[I3] BYRNES, C.; AND STEIG, D., "File management systems: a current summary," *Datamation* 15, 11 (Nov. 1969).

[I4] OLLE, T. W., "MIS: data bases," *Datamation* 16, 15 (Nov. 1970).

[I5] DIXON, PAUL, "The role of data management in management information systems," *IAG Journal* 3, 2 (August 1970).

[I6] CODASYL SYSTEMS COMMITTEE, "Introduction to 'feature analysis of generalized data-base management'," *Comm. ACM* 14, 5 (May 1971), 308-318.

(M) Data Models—Theory

[M1] CODASYL DEVELOPMENT COMMITTEE, "An information algebra, phase I report of the Language Structure Group," *Comm. ACM* 5, 4 (April 1962), 190-204.

[M2] CHILDS, D. L., "Feasibility of a set-theoretic data structure: a general structure based on a reconstituted definition of relation," *Proc. IFIP Congress 1968*, North-Holland Publ. Co., Amsterdam, The Netherlands, 1968, pp. 420-430.

CHILDS, D. L., "Description of a set-theoretic data structure," *Proc. AFIPS Fall Jt. Computer Conf.*, 1968, AFIPS Press, Montvale, N.J., 1968, pp. 557-564.

[M3] CODD, E. F., "A relational model of data for large shared data banks," *Comm. ACM* 13, 6 (June 1970), 377-87.

[M4] HABBGRAVE, W. T., "A technique for implementing a set processor," *Proc. ACM SIGMOD/SIGPLAN Conf. on Data Abstraction Definition and Structure*, 1976, Taylor and Ledgard, (Eds.).

(PL) Programming Languages

[PL1] CLIPPINGER, R. F., "FACT a business compiler description and comparison with COBOL and commercial translator," in *Annual Review in Automatic Programming* 12, 1, Pergamon Press, pp. 231-292.

[PL2] INTERNATIONAL BUSINESS MACHINES, *General Information Manual—IBM Commercial Translator*, Form F28-8043, IBM Corp., 1960.

[PL3] GENERAL ELECTRIC, "GECOM—the general compiler," CPI3 144 (IOM-4-011), General Electric Corp. Computer Dept., April 1961.

[PL4] PERSTEIN, M. H., *The JOVIAL J3 Grammar and Lexicon*, Tech Memo No. TM555/012/04, System Development Corp., Santa Monica, Calif., 1965.

[PL5] SAMMET, J. E., "Base elements of COBOL 61," *Comm. ACM* 6, 5 (May 1962), 237-253.

[PL6] CODASYL DATABASE TASK GROUP, "COBOL extensions to handle data bases," (PB 177 892), Jan. 1968.

[PL7] COBOL, *American National Standard Programming Language* COBOL, X3.23—1974, American National Standards Institute, Inc., New York, 1974.

(S) DBMS Specifications

[S1] CODASYL DATA BASE TASK GROUP, *October, 1969 Report*, (superseded by 1971 DBTG Report, currently out of print).

[S2] CODASYL DATA BASE TASK GROUP, *April 1971 Report*, (available from ACM).

[S3] CODASYL DATA DESCRIPTION LANGUAGE COMMITTEE, *CODASYL Data Description Language Journal of Development*, (June 1973), NBS Handbook 113, (Jan. 1974).

(SL) Stored-Data Definition

[SL1] SMITH, D. C. P., "An approach to data description and conversion," PhD Thesis, Moore School Report 72-20, Univ. of Pennsylvania, Philadelphia.

[SL2] CODASYL STORAGE STRUCTURE DEFINITION LANGUAGE TASK GROUP, *Design Objectives for a Storage Structure Definition Language*, 1970. (See [SL5]).

[SL3] STORAGE STRUCTURE DEFINITION LANGUAGE TASK GROUP (SSDLTG) of CODASYL Systems Committee,
FRY, J. P. "Introduction to storage structure definition."
MCGEE, W. C. "Informal definitions for the development of a storage structure definition language."
YOUNG, J. W., JR., "A procedural approach to file translation."
STIBLEY, E.; AND TAYLOR, R. "Preliminary discussion of a general data to storage structure mapping language," *Proc. 1970 ACM SIGFIDET Workshop on Data Description and Access*, pp. 368-380.

[SL4] TAYLOR, R. W., "Generalized data-base management system data structures and their mapping to physical storage," PhD Thesis, Univ. of Michigan, 1971.

[SL5] FRY, J. P.; SMITH, D. C. P.; AND TAYLOR, R. W., "An approach to stored-data definition and translation," *Proc. of 1972 ACM SIGFIDET Workshop on Data Description Access and Control*, pp. 13-55.

[SL6] BACHMAN, C. W., "The evolution of storage structures," *Comm. ACM* 15, 7 (July 1972), 628-634.

[SL7] SIBLEY, EDGAR H.; AND TAYLOR, ROBERT W., "A data definition and mapping language," *Comm. ACM* 16, 12 (Dec. 1973), 750-759.

(T) DBMS Texts

[T1] MARTIN, J., *Computer data-base organization*, Prentice-Hall, Englewood Cliffs, N.J., 1975.

[T2] DATE, C. J., *An introduction to data-base systems*, Addison-Wesley, Reading, Mass., 1975.

[T3] CAGAN, C., *Data management systems*, Melville Publ. Co., Los Angeles, Calif. 1973.

[T4] LEFKOVITZ, D., *Data management for on-line system*, Hayden, Publ. Co., Rochelle Park, N.J., 1974.

(U) Surveys

[U1] MINKER, JACK; AND SABLE, JEROME, "File organization and data management," in *Annual review of information science and technology*, Vol. 2, John Wiley & Sons, New York, 1967, pp. 185-222.

[U2] CODASYL Systems Committee, *A survey of generalized data base management systems*, (PB 203142), May 1969.

[U3] FRY, J. P., et al., "Data management systems survey," MITRE Corporation Report, MTP 329, (AD 684 907) Jan. 1969.

[U4] MINKER, J., "Generalized data management systems—some perspectives," Univ. of Maryland Computer Science Center Technical Report, Dec. 1969.

[U5] FRY, J. P., AND GOSDEN, J. A., "Survey of management information systems and their languages," in *Critical factors in data management*, F. Gruenberger, (Ed.), McGraw-Hill Book Co., 1969, pp. 41-55.

[U6] GUIDE INT., "Comparison of data-base management systems," Oct. 1971.

[U7] CODASYL SYSTEMS COMMITTEE, *Feature analysis of generalized data base*

Management Systems, May 1971, (available ACM).

[U8] Kozar, G. J., et al., "Data management systems catalogue," The MITRE Corp., Technical Report MTP 130, Jan. 1973, (available from The MITRE Corp).

(V) Vendor Systems

Applications Software, Inc.
Corporate Offices
21515 Hawthorne Boulevard
Torrance, Calif. 90503

[V1] Sukdeen, D. H., *AS-IST—a general purpose management system*, Application Software, Inc., San Pedro, Calif., 1968.

Burroughs Corporation
Burroughs Place
Detroit, Mich. 48232

[V2] Burroughs Corporation, "B6700/B 7700 DMS II data and structure definition language (Dasdl) reference manual," April 1974.

Burroughs Corporation, "B6700/B7700 DMS II host language interface reference manual," April 1974.

Cincom Systems, Inc.
2300 Montana Avenue
Cincinnati, Ohio 45211

[V3] Cincom Systems, "Total/7 reference manual: application programming," Pub. #P02-1321-2, June 1974.

Cincom Systems, "Total/7 reference manual—data base administration," Pub. #P02-1322-2, June 1974.

Computer Corporation of America
575 Technology Square
Cambridge, Mass. 02139

[V4] Computer Corporation of America, "CCA 204 data base management software system user language reference manual," March 1974.

Control Data Corporation
8101 34th Avenue South
Minneapolis, Minn. 55420

[V5] Control Data Corporation, Mars VI *reference manual*, Pub. # 17305100, CDC, 1974.

Control Data Corporation, Mars VI *reference manual (full inversion)*, Pub. # 17313000.

Control Data Corporation, Mars VI *reference manual (partial inversion)*, Pub. # 60385000.

[V6] Control Data Corporation, "Query update version 2.0 reference manual," Pub. # 60307500.

Control Data Corporation, "Data definition language for query/update subschema," Pub. # 60359200.

Cullinane Corporation
One Boston Place
Boston, Mass. 02108

[V7] Cullinane Corporation, "Integrated database management system program and reference."

Digital Equipment Corporation
146 Main Street
Maynard, Mass. 01754

[V8] DECsystem 10, "Data base management system programmer procedures manual," DEC-10-APPMA-B-D, 2d ed.

DECsystem 10, "Data base management system data base administration procedures manual," DEC-10-AAPMA-B-D, 2d ed.

Honeywell Information Systems
200 Smith Street
Waltham, Mass. 02154

[V9] I-D-S/II Related Publications:
I-D-S/II programmer reference manual, DE00.
I-D-S/II data base administrator guide, DE10.
Interactive I-D-S/II reference manual, DE11.
Ufas (United File Access System), DC89.
I/O supervisor, DD82.
File management supervisor, DD45.

[V10] Management Data Query System (MDQS):
MDQS, User Guide, DC80.
MDQS Data Base Administrator Guide, DC81.
MDQS IV, DD92.
MDQS IV Administrator Guide, DD94.

[V11] Dressen, P. C., "The dataBasic language—a data processing language for non-professional programmers," *Proc. AFIPS Spring Jt. Computer Conf.*, 1970, AFIPS Press, Montvale, N.J., 1970, pp. 307–312.

Informatics
Mark IV Systems Co.
21050 Vanowen Street
Canoga, Calif. 91303

[V12] Postley, J. A., "The Mark IV system," *Datamation*, 14, 1 (Jan. 1968), 28–30.

IBM (for IBM information, see local representative)

[V13] *Information Management System/360 (IMS/360) application description manual*, IBM Form No. H20-0524.

[V14] *Information Management System/360 Version 2, general information manual*, IBM Form No. GH20-0765.

[V15] *Information Management System Virtual Storage (IMS/VS) general information manual*, IBM Form No. GH20-1260.

[V16] *Generalized information system application description manual*, IBM Form No. GH20-0179.

[V17] Bryant, J. H.; and Semple, P., "GIS and file management," *Proc. ACM 1968 National Conf.*, ACM, New York, N.Y., 1968, pp. 97–107.

MRI Systems Corporation
Box 9968
Austin, Texas 78766

[V18] System 2000 Publications:
General information manual, G-1.
Basic *reference manual* (Includes binder), A-1.
Immediate access feature, I-1.
Cobol *procedural language interface feature*, C-1.
Fortran *procedural language interface feature*, F-1.
PL/1 *procedural language interface feature*, P-1.
Report writer feature, R-1.

Philips-Electrologica, B. V.
PO Box 245
Apeldoorn, The Netherlands

[V19] Publications:
Introduction to Pholas, Pub. no. 5122 991 25221.
Pholas *sub-schema DDL and SML*, Pub. no. 5122 991 25861.
Pholas *schema DDL and SSL*, Pub. no. 5122 991 25841.

Shipping Research Services, Inc.
205 S. Whiting Street
Alexandria, Va. 22304

[V20] Shipping Research Services, Inc., "The data base system Sibas: an introduction," 1974.
Aschim, F. F.; and Boone, P., "Sibas—an implementation of the CODASYL data base concept," *Management Informatics* 2, 3 (1973).

Software ag
Reston International Center
11800 Sunrise Valley Drive
Reston, Va. 22091
61 Darmstadt
Hilpertstrasse 20
West Germany

[V21] Adabas *introduction*; Adabas *reference manual*; and Adabas *utilities manual*, Software ag of North America, Reston, Va.

Sperry Univac
PO Box 500
Blue Bell, Pa. 19422
ATV House
17 Great Cumberland Place
London W1, England

[V22] "Univac 1100 Series, Data Management System (DMS 1100) schema definition, data administrator reference," Sperry Rand Corp., 1972, 1973.
"Univac 1100 Series, Data Management Systems (DMS 1100) American National Standard Cobol (Fieldata), data manipulation language, programmer reference," Sperry Rand Corp., 1972.

Xerox Corporation
701 South Aviation Boulevard
El Segundo, Calif. 90245

[V23] "Xerox Extended DMS Sigma 6/7/9," Reference Manual, 90301 2B, February 1974.

Set Theoretic Information Corporation
117 N. 1st Street
Ann Arbor, Mich. 48104

[V24] Set Theoretic Information Corporation, "STDS/I reference guide," 1975.

(W) Report Generator

[W1] "Share 7090 9Pac, Part I: "Introduction and general principles," in *7090 Programming Systems, Systems Reference Library*, IBM, File 7090-28, Form J28-6166-1, p. 32, 1961.

[W2] Leslie, H., "The report generator," *Datamation*, (June 1967), 26-28.

[W3] Friedberg, L. M., "RPG: the coming of ACE," *Datamation*, (June 1967), 29-31.

[W4] Longo, F., "Surge: a recording of the Cobol merchandise control algorithm," *Comm. ACM* 5, 2 (Feb. 1962), 98-100.

(X) DBMS prior to 1968

[X1] Miller, L.; Minker, J.; Reed, W.; and Shindle, W., "A multi-level file structure for information processing," *Proc. Western Jt. Computer Conf.*, 1960, Spartan Books, New York, 1960, pp. 53-59.

[X2] Green, B. V.; Wolf, A. K.; Chomsky, C.; and Laughery, J., "Base-ball, an automated question-answer," *Proc. Western Jt. Computer Conf.*, May 1961, Spartan Books, New York, 1961.

[X3] Vesper, N. R., *Information Retrieval Program*, Report C-1210 (David Taylor Model Basin), May 1961.

[X4] Postley, J. A.; and Bunyell, T. D., "Generalized information retrieval and listing system," *Datamation* 4, 12 (Dec. 1962), 22-26.

[X5] *User's Manual for TUG-Format Table Tape Updater and Generator*, Naval Command Systems Support Activity, prepared by International Business Machines Corp., Rockville, MD. Oct. 1962.

[X6] Collilla, R. A.; and Sams, B. H., "Information structure for processing and retrieving," *Comm. ACM* 5, 1 (Jan. 1962), 11-15.

[X7] Cheatham, T. E., Jr.; and Warshall, S., "Translation of retrieval requests couched in a 'semiformal' English-like language," *Comm. ACM* 6, 1 (Jan. 1962), 34-39.

[X8] Clemenson, W. D., "Recol—a retrieval command language," *Comm. ACM* 6, 3 (March 1963), 117-122.

[X9] Naval Command Systems Support Activity, "User's manual for the 704/7090 information retrieval," NAVCOSSACT Document No. 10S001, CM-70, Nov. 1963.

[X10] *Intelligence Data Processing System Formatted File System*, U.S. Navy Fleet Intelligence Center and Intelligence Systems Dept. IBM Federal Systems Div., May 1963.
Vol. 1, *Program description*
Vol. 2, *Program flowdiagrams and listings*

Vol. 4. *Information system design and utilization*
Vol. 5. *Information retrieval.*

[X11] Naval Command Systems Support Activity, "User's manual for NAVCOSSACT information processing system phase I library maintenance system," NAVCOSSACT Document No. 88M008, CM-52, August 1963.

[X12] Naval Command Systems Support Activity, "User's manual for NAVCOSSACT information processing system phase I," NAVCOSSACT Document No. 90S003A, CM-51, July 1963. Supplement 1 published Jan. 1964.

[X13] System Development Corp., "System design specifications for Lucid phase 1," Tech. Memo No. TM-1749/000/00, Santa Monica, Calif., Jan. 1964.
Vol. 1. Lucid control system design
Part 1. *The Master Tape*, Tech Memo No. TM-1749/101/00.
Part 2. *Parameter Load*, Tech Memo No. TM-1749/102/00.
Part 3. *Operational Control*, Tech Memo No. TM-1749/103/00.
Part 4. *Test Set-Up*, Tech Memo No. TM-1749/104/00.
Vol. 2. "Gendarme data processing facilities," Tech. Memo No. TM-1749/201/00.
Vol. 3. "Lucid program design: the grammar of Opague," Tech. Memo No. TM-1749/301/10.

[X14] Bryant, J. H., "Aids experience in managing data-base operation," *Proc. of the Symposium on Development and Management of a Computer-Centered Data Base*, A. Walker, (Ed.), System Development Corp., Santa Monica, Calif., 1964, pp. 36–42.

[X15] Bachman, C. W.; and Williams, S. B., "A general purpose programming system for random access memories," *Proc. AFIPS Fall Jt. Computer Conf.*, 1964, Vol. 26, Spartan Books, New York, 1964, pp. 411–422.

[X16] Naval Command Systems Support Activity, "User's manual 1401 Tuff tape updater for formatted files," NAVCOSSACT Document No. 90S012W, CM-108, May 1964.

[X17] NMCS Information Processing System (Nips), IBM 1410, NMCS Support Center, Washington, D.C., 1964.

[X18] Integrated Data Store—A New Concept in Data Management, Publication CPB-453 (5C10-16), General Electric Co.

[X19] Naval Command Systems Support Activity, "7090 information processing system revised," NAVCOSSACT Document No. 90M002, 0M-01, Oct. 1965.

[X20] Naval Command Systems Support Activity, "User's manual for 704/7090 Tuff Mop III tape updater for formatted files," NAVCOSSACT Document No. 10S001, CM-74, Nov. 1963. Change 1 published Feb. 1964. Change 2 published August 1965.

[X21] Grant, E., Lucid *User's Manual*, Tech Memo No. TM-2354/001, System Development Corp., Santa Monica, Calif. June 1965.

[X22] Spitzer, J. F., et al., "The Colingo system' design philosophy," in *Information System Sciences, Proc. of the Second Congress*, 1965, Spartan Books, New York, 1965, pp. 36–39.

[X23] SDS MANAGE Reference Manual, Publication 90-10-46A, Scientific Data Systems, May 1966.

[X24] Connors, T. L., "Adam—a generalized data management system," *Proc. AFIPS Spring Jt. Computer Conf.*, 1966, Vol. 28, Spartan Books, New York, 1966, pp. 193–203.

[X25] A User's Guide to the Adam System, MTR-265, MITRE Corp., (AD 664 332), August 1966.

[X26] IDHS 1410 Formatted File System: File Maintenance and File Generation Manual, Defense Intelligence Agency, DIAM-65-9-1, August 1966. Also, IDHS 1410 Formatted File System: Retrieval and Output Manual, DIAM-69-9-2.

[X27] A Description of the Internal Operations of the Adam System, MTR-216, MITRE Corp., (AD 660 581), August 1966.

[X28] Dodd, G. G., "APL—a language for associative data handling in PL/1," *Proc. AFIPS Fall Jt. Computer Conf.*, 1966, Vol. 29, Spartan Books, New York, 1966, pp. 667–684.

[X29] Vorhaus, A.; and Mills, R., *The Time-Shared Data Management System: A New Approach to Data Management*, Tech Memo SP-2747, System Development Corp., Santa Monica, Calif. 1967.
Williams, W. D.; and Bartram, R. C., *COMPOSE/PRODUCE: A User-Oriented Report Generator Capability Within the SDC Time-Shared Data Management System*, Tech Memo SP-2634, System Development Corp., Santa Monica, Calif. 1967.

[X30] Steil, G. P., "File management on a small computer," *Proc. 1967 AFIPS Spring Jt. Computer Conf.*, Spartan Books, New York, 1967, pp. 199–203.

[X31] Dixon, Paul J.; and Sable, J., "DM-1—A generalized data management system," *Proc. AFIPS Spring Jt. Computer Conf.*, (30), 1967, 185–198.

[X32] Naval Command Systems Support Activity, "User's manual for information processing system phase 3A for the AN/FYK-1 (V) data processing set," NAVCOSSACT Document No. 88M001A, CM-123, Revision 5, August 1967.

[X33] AFLC/ESD/MITRE, *Advanced Data Management* (Adam) *Experiments*, Final Report, (AD 648 226), Feb. 1967.

[X34] Brown, R.; and Nordyke, G. P., "ICS an information control system," *Proc. IFIPS Conf. Mechanized Information Storage, Retrieval and Dissemination*, 1967, North-Holland Publ. Co., Amsterdam, The Netherlands, 1967.

[X35] *Data Language No. 1 (DL-1) Encyclopedia Pub. 2841-F*, North America Aviation,

Inc., and International Business Machines Corp., 1967.

[X37] GILDEA, R. A., *Evaluation of* ADAM *an advanced data management system*, MITRE Corp., (AD 661 273), May, 1967.

(Y) DBMS 1968 to Present

[Y1] BLEIER, R. E., *Treating Hierarchical Data Structures in the SDC Time-Shared Data Management System (TDMS)*, Tech Memo Sp-2750, System Development Corp., Santa Monica, Calif., 1968.

[Y2] BLEIER, R. E.; AND VORHAUS, A., *File Organization in the SDC Time-Shared Data Management System (TDMS)*, Tech Memo SP-2750, System Development Corp., Santa Monica, Calif., 1968.

[Y3] RAUCHER, V.; AND SCHWIMMER, H. S., *The Time-Shared Data Management System (TDMS), Language Specifications*, Tech Memo TM-3370, Systems Development Corp., Santa Monica, Calif., 1968.

[Y4] *SDS/9 SERIES Manage*, Publication No. CB 10035, Scientific Data Systems, 1968.

[Y5] ATLEE, E. S., et al., "COGENT III functional specifications," Computer Sciences Corporation, 1968.

[Y6] WELSH, W. A., "Engineered design of EDP systems," *Systems and Procedures Association Internal Meeting*, October 1968.

[Y7] "Remote file management system (RFMS), "*Computation Center Technical Staff Documentation*, Publications 0 to 14, Univ. of Texas at Austin, 1968.

[Y8] MANGOLD, C. A., "COBOL data management system (CDMS) briefing," *Proc. Guide*, 30, (May 1970), 175-720.

[Y9] MCELROY, D. C., "The SERIES data management system," *Datamation* 16, 4 (April 1971), 131-138.

[Y10] NAVAL COMMAND SYSTEMS SUPPORT ACTIVITY, "Information processing system (IPS) user's guide," NAVCOSSACT Document No. 88M1904, TR-03, September 1971. Change 1 published Feb. 1972. Change 2 published Feb. 1973.

[Y11] MEINERS, E. E., "A machine-independent data management system" *Datamation*, 19, 6 (June 1973), 92-98.

[Y12] NMCS INFORMATION PROCESSING SYSTEM 360 FORMATTED FILE SYSTEM (NIPS FFS), NMCS Support Center, CSM UM 15-74 October 1974).

Vol. I: Introduction to file concepts.
Vol. II: File structuring (FS).
Vol. III: File maintenance (FM).
Vol. IV: Retrieval and sort Processor (RASP).
Vol. V: Output processor (OP).
Vol. VI: Terminal processing (TP).
Vol. VII: Utility support (UT).
Vol. VIII: Job preparation.
Vol. IX: Error codes.
TR 54-74: Installation of NIPS 360 FFS.

(Z) Relational Systems

[Z1] GOLDSTEIN, R. C.; AND STRNAD, A. L., "The MACAIMS data management system," *Proc. 1970 ACM-SIGFIDET Workshop on Data Description and Access*, Nov., 1970, pp. 201-229.

[Z2] MCINTOSH, S.; AND GRIFFEL, D., "Data management for a penny a byte," *Computer Decisions*, (May 1973).

[Z3] WHITNEY, V. K. M., "RDMS: a relational data management system," *Proc. Fourth Internatl. Symposium on Computer and Information Sciences (COINS IV)*, Dec. 1972, Plenum Press, New York, 1973

AVAILABILITY OF REFERENCES

Addresses	Publications
EDP Analyzer Canning Publications, Inc. 925 Anza Avenue Vista, Calif. 92083	EDP Analyzer
ACM Association for Computing Machinery 1133 Avenue of the Americas New York, N.Y. 10036 (212) 265-6300	SIGBDP DBTG Specifications CODASYL Systems Committee Report SIGMOD Proceedings SIGFIDET Proceedings *Comm. ACM* *JACM* *TODS* Very Large Data Base Proceedings
Management Information Systems Research Center Graduate School of Business Administration University of Minnesota Minneapolis, Minn 55455	MISRC Publications
IFIP Administrative Data Processing Group 6 Stadhouderskade Amsterdam 1013, The Netherlands	CODASYL System Committee DBTG Specification *IAG Journal*

Addressee	Publications
Technical Services Branch Department of Supply and Services 86 Metcalfe Street Fifth Floor Ottawa, Ont., Canada K1A 0S5	CODASYL COBOL Specification
British Computer Society 29 Portland Place London W1, England	CODASYL System Committee DBTG Specifications
National Technical Information Service 5285 Port Royal Road Springfield, Va. 22151	Documents with AD or PB numbers
SHARE Inc. One Illinois Center 111 E. Wacker Drive Suite 600 Chicago, Ill. 60601	SHARE Proceedings
GUIDE Int. Mr. Sandy Hill Smith, Bucklin, and Associates 111 E. Wacker Drive Chicago, Ill. 60601	GUIDE Proceedings
System Development Corp. 2500 Colorado Boulevard Santa Monica, Calif.	SDC Technical Reports, Memorandums
The MITRE Corp. Bedford Operations Box 207 Bedford, Mass. Washington Operations Westgate Research Park McClean, Va. 22101	MITRE Technical Reports

centro de educación continua
división de estudios de posgrado
facultad de ingeniería unam

CASOS PRACTICOS DEL DISEÑO DE SISTEMAS DE INFORMACION

LA COMPUTACION DISTRIBUIDA COMO UNA ALTERNATIVA DEL FUTURO

DR. ADOLFO GUZMAN ARENAS

AGOSTO, 1980

LA COMPUTACION DISTRIBUIDA COMO UNA ALTERNATIVA DEL FUTURO

ADOLFO GUZMAN ARENAS
Departamento de Computación, IIMAS
Universidad Nacional Autónoma de México

Se presentan, definen y describen diferentes tipos de sistemas distribuidos. Se introduce el concepto de dator. Se describen tres proyectos de Sistemas Distribuidos que están siendo realizados en el Depto. de Computación (IIMAS) de la Universidad Nacional Autónoma de México.

El artículo lanza una serie de arquitecturas que anteriormente habían sido propuestas o pensadas en diferentes lugares, pero que no habían sido asociadas explícitamente a procesamiento distribuído.

RESUMEN

Después de haber fabricado enormes computadoras que son usadas a través del tiempo compartido y la multiprogramación, las economías de la integración en gran escala inducen al hombre a pensar en multitud de máquinas modestas (microcomputadoras) que colaboran para desarrollar las mismas atereas que hasta ahora han venido desarrollando sus hermanas mayores.

Es la división y repartición del trabajo en múltiples unidades de procesamiento; es la intercomunicación, colaboración y coordinación de tales unidades; es el abaratamiento de la transmisión de datos, al solo transmitirse información requerida; es el procesamiento local de la información local; éstos son los atributos principales de sistemas de procesamiento distribuido, los que son analizados en este artículo.

También se mencionan las redes de computadoras, antecesores ya clásicos de la computación distribuida.

El artículo concluye con la discusión de tres proyectos de omputación distribuida que se están realizando en el Departamento de Computación en la UNAM: (1) la construcción de una computadora "grande" que posee LISP como lenguaje de máquina, y que internamente está formada por docenas de microprocesadores; (2) la construcción

de una máquina especialmente diseñada para manejo eficiente de bases de datos y utilizable en el soporte de sistemas de información, (3) un sistema de procesamiento distribuido que, mediante una repartición juiciosa de la carga de procesamiento, y a través de una conexión (cable coaxial) de alta velocidad, elimina la necesidad de tener una computadora "grande" en el sistema, el que únicamente consta de la red de "terminales" inteligentes.

PROBLEMAS PROPIOS DE COMPUTACION CENTRALIZADA

Para aplicaciones que no necesitan extensivamente capacidad de cómputo, la gran computadora puede ser costosa. Los costos de telecomunicaciones pueden ser excesivos en una organización con usuarios dispersos geográficamente. Una instalación local pudiera no tener todos los recursos requeridos para un trabajo específico. Los tiempos de respuesta para trabajos y procesos en tiempo real pueden tornarse impredecibles e incontrolables debido a la gran variación en carga típica de una instalación de múltiples usuarios. La gran complejidad del hardware y del software del sistema operativo necesarios en una gran instalación pueden fallar más de lo aceptable, frente a ciertos trabajos con tiempos críticos de entrega. Un usuario puede recibir poca atención y servicio deficiente de un gran centro de cómputo, debido a su gran dimensión y compleja estructura.

LA ECONOMIA DE ESCALAS AHORA FAVORECE AL PEQUEÑO COMPUTADOR

Los grandes y rápidos cambios en la tecnología de semiconductores han tenido un impacto fuerte en los costos relativos de las diferentes partes de un sistema de cómputo que posee comunicaciones. Mientras que los costos de lógica y de almacenamiento interno han disminuido muy rápidamente, los costos de almacenamiento (memoria) masivo han disminuído menos rápidamente, en tanto que los costos de comunicación han permanecido prácticamente constantes. Además, el software se ha convertido en un porcentaje creciente del costo total del sistema de cómputo. Ya no hay, pues, una penalidad económica severa en contra de distribuir la capacidad de procesamiento a las terminales individuales.

En sistemas de tiempo compartido y de multiacceso, hay varias razones sólidas funcionales para centralizar la base de datos, particularmente en aquellos casos donde varios usuarios pueden alterarla y accesarla. Sin embargo, no existen fuertes razones funcionales para centralizar las funciones mismas de procesamiento. La decisión de centralizar el procesamiento del usuario individual es una decisión económica. Con las tendencias actuales en los costos de equipo digital por una parte y los de comunicación por la otra, es económicamente atractivo efectuar tanto como sea posible de cómputo y procesamiento localmente, para así minimizar las necesidades de transmisión de datos y por ende los costos de comunicación

VENTAJAS DE COMPUTACION DISTRIBUIDA.

Puesto que el costo ya no es una barrera, se pueden aplicar las ventajas inherentes de los sistemas distribuidos a una gran cantidad de problemas de cómputo. Estas ventajas incluyen las siguientes:

1. Confiabilidad. Se puede tener redundancia de una manera relativamente barata, en un sistema distribuido. El sistema entero no tiene que ser duplicado, como es el caso de una sola computadora. Solo un número pequeño de procesadores deben ser agregados para asegurar el grado requerido de disponibilidad. Se pueden conseguir estructuras de software más simples, y por lo mismo más confiables, en una colección de pequeñas computadoras.

Capacidad de cómputo a bajo costo

Se presenta la capacidad de cómputo barata, en la forma de minicomputadoras, como una alternativa atractiva frente a las computadoras medianas y grandes, para el desfogue de muchos trabajos de cómputo. Esta tendencia se incrementará en el futuro. Aún más, puede argüirse que la economía de escalas favorece ahora al computador pequeño. El costo por unidad de potencia de cómputo es menor en una línea de ensamble de un gran número de pequeñas computadoras, que en una línea de ensamble de un pequeño número de grandes computadoras.

2. Responsividad. El sistema distribuido puede ser más rápido en sus reacciones (más responsivo), puesto que se puede proporcionar acceso directo a una computadora, aún a pequeñas comunidades de usuarios. Esta responsividad toma la forma de un tiempo de ida—y—vuelta reducido en ambientes de procesamiento por lotes, y significa tiempos de respuesta más rápidos en un ambiente de teleproceso o tiempo real.

3. Crecimiento incremental. Un sistema distribuido próximo a sobrecargarse puede ser expandido incrementalmente a bajo costo, mediante la adición de más procesadores. Un sistema centralizado próximo a sobrecargarse puede asimismo ser descargado mediante la distribución de ciertas funciones hacia procesadores pequeños.

4. Correspondencia con patrones organizacionales. Muchas organizaciones poseen una naturaleza descentralizada. Podría presentar restricciones no naturales para la operación eficiente de la organización el tener instalaciones de cómputo centralizadas. Esta restricción se alivia mediante computación dispersa, pues permite a un grupo descentralizado su acceso a su propia computadora, la que puede optimizarse para las necesidades de tal grupo. Un arreglo en forma de red permitirá a la gerencia central tener acceso a información resumida de todos los grupos.

5. Se comparten recursos. Una red de sistemas de cómputo distribuidos permite a los usuarios en un sitio usar ventajosamente los recursos que se encuentran disponibles en otras localidades. Estos recursos pueden consistir de programas, bases de datos, y poder de cómputo. Las redes que comparten recursos permiten balance de carga (de cómputo), respaldo, y disminuyen la duplicación de esfuerzos.

PROBLEMAS DE LA COMPUTACION DISTRIBUIDA

1. Algunos tipos de problemas se resuelven sobre todo en sistemas grandes, por ejemplo aplicaciones numéricas.

2. Un diseño pobre puede conducir a una pérdida de confiabilidad y de responsividad.

3. La descentralización de la capacidad de cómputo puede conducir a un crecimiento incontrolado, a imperios locales, y a la difusión y dispersión de personal de cómputo dentro de la organización.

4. Un sistema conteniendo muchas computadoras es bastante complejo de construir y validar. Aún no existen disponibles en el mercado una gran variedad de software que conduzca a la computación distribuida.

Antes de tomar decisión alguna, deben pesarse cuidadosamente las ventajas y desventajas de procesamiento centralizado ver su procesamiento distribuido. Sin embargo, hay razones para creer que la tendencia hacia procesamiento distribuido crecerá, particularmente conforme el costo de poder de cómputo decline.

DIFERENTES DEFINICIONES DE SISTEMAS DISTRIBUIDOS

A. Para algunos usuarios, un sistema distribuido es un conjunto de terminales inteligentes, situadas en el lugar donde se requiere primariamente su uso, para dar a los elementos de la organización local más poder de cómputo y un soporte de respuestas más rápida. Estas terminales llevan a cabo la mayoría de las funciones de cómputo para el grupo local. Cuando se torna necesario, las terminales se comunican entre sí y con grandes computadoras remotas, para mejor servicio.

Estos usuarios están preocupados por la optimización de sus funciones dentro del negocio, determinación de los mejores productos disponibles para la construcción del sistema, reducción de costos de comunicación, y tal vez con problemas severos de control asociados con facilidades de cómputo descentralizadas.

B. Para otros usuarios, un sistema distribuido es una colección de múltiples computadoras o elementos de procesamiento, trabajando estrechamente vinculados en la solución de un solo problema. Un ejemplo podría ser un sistema en línea para Bancos, constituido por varias minicomputadoras ligadas entre sí por memoria compartida, por líneas de comunicación y/o por vías (buses) comunes. Cada minicomputadora procesa un subconjunto de las transacciones bancarias y actualiza una porción de una base de datos común.

Los usuarios de tales sistemas están preocupados con problemas de diseño de hardware y software, confiabilidad, sistemas operativos para multicomputadoras, y cómo descomponer óptimamente programas y bases de datos. También se preocupan de la integridad de la base de datos, y de la forma de poder seguir efectuando transacciones (y brindando un servicio adecuado) frente a fallas de partes del sistema: terminales, líneas de transmisión, conmutadores de mensajes, etc.

Limitaciones en el Desarrollo de Sistemas de Cómputo.

En el presente, los principales factores que limitan las mejoras ulteriores en todos tipos de sistemas de computación, incluyendo sistemas en línea, son:

1. *Equipo de entrada y salida. Velocidad, costo y confiabilidad.*

2. *Memoria. Su velocidad y costo, tanto de memoria periférica como de memoria principal (interna).*

3. *Software. Su costo, su complejidad, el almacenamiento requerido, la dificultad de convertirlo a otra máquina (a otro hardware).*

C. También hay usuarios para los cuales procesamiento distribuido significa el uso de pequeñas computadoras para desarrollar funciones que anteriormente habían sido desarrolladas por computadoras grandes.

Estas pequeñas computadoras, al absorber tales funciones, relevan de carga a las máquinas grandes y prolongan su vida. Por ejemplo, procesadores frontales de comunicaciones se han venido usando durante años para manejar comunicaciones de datos. Recientemente, se han desarrollado computadores subalternos (backend) para dar soporte a funciones de administración de bases de datos. Los diseñadores de tales sistemas se preocupan de los métodos para acoplar máquinas pequeñas a grandes, interacciones de los sistemas operativos, y el desarrollo de programas eficientes para los sistemas de soporte.

El denominador común de los sistemas distribuidos

El tema común que liga los diferentes tipos de sistemas distribuidos es el uso de múltiples elementos de cómputo que colaboran entre sí para realizar trabajos que hasta ahora habían sido desarrollados en grandes computadoras, o no habían sido desarrollados en ningún lado.

Los sistemas distribuidos se han desarrollado como alternativas a grandes instalaciones de cómputo porque, en muchas circunstancias, las instalaciones centrales no han sabido proveer soluciones óptimas o efectivas a problemas importantes.

D. Otros usuarios entenderán por sistema distribuido colecciones de centros de cómputo, geográficamente dispersos e independientes, ligados entre sí por líneas de transmisión de datos para permitir el uso común y compartido de recursos de hardware y de software. Estos sistemas se conocen como redes de cómputo, y son los predecesores de los veradaderos sistemas de procesamiento distribuido.

Un ejemplo es una red desarrollada por varias instituciones educativas, cada quien con su propia instalación de cómputo. La red permite a un usuario en una instalación el acceso a un programa o a un archivo residente en otro centro de cómputo. La red puede permitir asimismo la nivelación de las cargas y el respaldo cuando algún integrante de la red falla.

Los diseñadores de tales redes se preocupan por la seguridad de la red, el mantenimiento de la independencia local, algoritmos de cobranza apropiados, administración de la red, comunicaciones óptimas, y el eslabonamiento de computadoras heterogéneas.

PRINCIPALES CATEGORIAS DE SISTEMAS DE PROCESAMIENTO DISTRIBUIDO

En razón del grado de acoplamiento entre los elementos de procesamiento y el propósito para el cual se desarrolló el sistema, se distinguen tres categorías de sistemas de procesamiento distribuido:

(1) débilmente acoplados, cuando los procesadores están conectados por líneas seriales de ancho de banda relativamente bajo (inferior a 50 kilobits/seg). En su mayoría son redes de computadoras que permiten compartir recursos. Estas redes conectan centros de cómputo de propósito general que se encuentran dispersos geográficamente, siendo la conexión de acoplamiento débil. La red permite que los usuarios en una localidad usen ventajosamente los recursos disponibles en otras localidades.

Cuando las computadoras son heterogéneas y operan bajo diferentes estructuras de administración, surgen problemas significativos de software y de administración en este tipo de redes.

En este tipo de sistemas una computadora está restringida a su "medio ambiente" local (periféricos y memoria), pues los vínculos con otras máquinas son lentos. No se comparte memoria principal, ni se ejecutan trabajos por dos máquinas diferentes de la misma memoria de acceso aleatorio.

Cuando las computadoras son heterogéneas, no hay nivelación de carga, por regla general.

(2) moderadamente acoplados, cuando los procesadores están conectados por líneas seriales de alta velocidad (superior a 50 kilobits/seg), vías (buses) paralelas, y/o periféricos compartidos. Estos sistemas se conocen generalmente como sistemas de múltiples procesadores. Normalmente están formados por procesadores homogéneos (todos son del mismo tipo), y el sistema está dedicado a un conjunto de trabajos limitados y bien definidos. Los procesadores pueden estar en la misma localidad o geográficamente dispersos.

Este tipo de sistemas, cuando está formado por minicomputadoras, poseen el potencial de rebasar y superar a máquinas grandes de uso general, más caras, en una diversidad de aplicaciones dedicadas.

Algunos sistemas moderadamente acoplados consisten de computadoras grandes auxiliadas por computadoras chicas para desfogar funciones críticas. Como ejemplo se puede citar el procesador subalterno (back-end) para manejo de bases de datos. Este es un proyecto que actualmente se encuentra en desarrollo en el Departamento de Computación (IIMAS) de las Universidad Nacional Autónoma de México: la manufactura de una máquina para manejar bases de datos eficientemente.

(3) fuertemente acoplados, cuando los procesadores están conectados por memoria principal compartida, vías de acceso directo a memoria (DMA) y velocidades en los millones de bytes/seg.

Los multiprocesadores fuertemente acoplados proporcionan el potencial para una alta confiabilidad, alto desempeño, y computación geográficamente centralizada. Un acoplamiento fuerte entre procesadores siempre es un reto tecnológico en la coordinación de procesadores, y en software con imaginación.

Más adelante se describe el proyecto AHR: la construcción de una máquina que procesa LISP como lenguaje de máquina, y que internamente está constituida por varias docenas de microprocesadores Z-80. Este es un ejemplo de un sistema de multiprocesadores fuertemente acoplados, y está siendo llevado a cabo en el Departamento de Computación (IIMAS) de la UNAM.

EL DATOR COMO CELULA DEL PROCESAMIENTO DISTRIBUIDO

Bases de datos distribuidos

Una base de datos distribuida está formada por dos o más elementos de procesamiento, cada uno de los cuales tiene un conjunto de archivos asociado a él. Estos archivos pueden estar divididos en razón geográfica, o debido a bases del funcionamiento de la compañía, o bien por razones de redundancia y confiabilidad.

Por regla general, una base de datos distribuida se construye sobre un sistema distribuido débil o moderadamente acoplado.

Un dator es la célula constituyente de un sistema de procesamiento distribuido, en especial de los débil y moderamente acoplados.

Un dator es un sistema de procesamiento/captura/almacenamiento de la información que posee las siguientes características:

1) capacidad de procesamiento propia,
2) interacción con el usuario. Este interacciona el dator con la computadora central, cuando ésta existe aún;
3) programas almacenados;
4) parte del procesamiento es local y lo realiza el dator;
5) comunicaciones en línea con otros datores, con la(s) computadora(s) central(es) y sus bases de datos correspondientes;
6) entrada de datos orientada a la interacción con el usuario: teclado, lápiz luminoso, digitalizador de mapas, etc.;
7) salida de datos orientada al usuario: impresora serial, tubo de rayos catódicos, sintetizador de voz, etc.;
8) habilidad para compartir el procesamiento de trabajos, y cierto grado de coordinación entre datores;
9) habilidad para compartir información con otros datores;
10) (opcional) habilidad para balancear cargas con otros datores;
11) almacenamiento local (v. gr., discos) de datos.

Las habilidades (1), (3) y (4) le dan al dator características de computadora; las habilidades (2), (6) y (7) le dan al dator características de terminal; las habilidades (4), (5), (8) y (9) le dan al dator propiedades para poder desarrollar computación distribuida; las habilidades (8), (10) y (9) hacen que los periféricos (con la información en ellos contenida) sean promiscuos; las habilidades (9) y (10) hacen que los datores (los elementos de procesamiento) sean promiscuos.

Los datores eliminan concentración de la capacidad de procesamiento en una computadora central, repartiéndosela entre ellos.

Los datores son terminales tan inteligentes que han eliminado (o reducido grandemente) la computadora central a la que estaban conectados.

QUE ES UN DATOR

Un dator es al mismo tiempo computadora, terminal, nodo de una red, concentrador de información, adquisidor de datos y expulsor/exhibidor de resultados.

Si existe, la computadora central de un sistema distribuído construído con datores tiene las siguientes funciones:

a) Mantiene una base de datos central;
b) provee capacidad de cómputo elevada para problemas que no pueden descuartizarse satisfactoriamente entre los datores constituyentes.

Si no existe, la computadora central de un sistema distribuído constituído con datores es reemplazada, con elevadas economías, por un conjunto de datores que colaboran en las diferentes tareas de la instalación: captura de datos, procesamiento de información, mantenimiento de una base de datos, generación y distribución de resultados.

CARACTERISTICAS DE LOS SISTEMAS DISTRIBUIDOS FORMADOS POR DATORES

Los sistemas distribuídos, cuando la distinción entre unidad central de proceso y terminal se desvanece, exhiben varias características deseables, además de las propias de los sistemas distribuídos (véase "Ventajas de Computación Distribuída", al principio de este artículo), entre las cuales aparecen:

Descentralización del procesamiento. Un sistema distribuído de datores puede poseer o no una (o más) computadoras centrales. La eliminación de esta(s) computadora(s) ofrece ahorros económicos, puesto que varios computadores pequeños (datores) son más baratos que uno grande (véase "La Economía de Escalas Ahora Favorece al Pequeño Computador", arriba).

También se ahorra el envío de información al lugar central de procesamiento, al procesarse la información localmente, en el lugar donde se produce, y por lo común por el dator mismo que la capta. Es decir, la computación o procesamiento está distribuída.

A veces, y especialmente frente a grandes problemas numéricos (v. gr., inversión de grandes matrices; problemas de programación dinámica o no lineal), no se sabe el algoritmo para obtener eficazmente la solución del problema usando datores distribuídos, y tiene que seguirse el procedimiento convencional de usar una computadora grande central. Sin embargo, ésta a veces puede reducirse a un uso especializado de "triturador numérico", y aún pudiera ser sustituída por procesadores especializados (procesadores para arreglos, para Transformadas Rápidas de Fourier, procesadores de punto flotante, etc.) En estos últimos casos, se tendría un computador central ajustado a la albor especial, auxiliado por datores en una configuración distribuída.

Nivelación de carga en redes

La habilidad de tomar una carga de trabajo dada y distribuirla entre las computadoras de una red o los datores de un sistema distribuído, para así hacer un uso igual de los recursos de la red, es un servicio común en un gran número de redes. Se ofrece en las redes CYBERNET, DCS y TSS, pero no es una característica básica de las redes ARPA, MERIT, OCTUPUS o TUCC. En éstos puede agregarse bajo ciertas restricciones.

Nivelación de cargas. Con esto se designa la distribución de los diferentes trabajos a los diferentes datores, de tal suerte que los recursos del sistema se usen equitativa o niveladamente.

Si consideramos que cada usuario está conectado a cada teclado activo entonces la nivelación de cargas en un sistema distribuído puede tomar tres aspectos:

I. Hay un solo proceso (task) por usuario (es decir, por teclado). En este caso, salvo que se sepa cómo efectuar paralelismo dentro de un mismo task (lo que ocurre por ejemplo en la Máquina de LISP del Proyecto AHR, a ser descrito después), no hay posibilidad de nivelar la carga, pues a cada usuario se le asigna cuando mucho una c.p.u. y no hay dos cpu's colaborando en el mismo proceso.

II. Hay varios procesos por usuario. Aún así, puede no haber nivelación de cargas, si (como es el caso de las computadoras PERQ de la compañía THREE RIVERS) se asigna una c.p.u. a todos los procesos del mismo usuario. En este caso, se procesan en la localidad donde el usuario reside los trabajos que él genera, por la c.p.u. local. El monitor local es de multiprogramación, ya que puede atender varios trabajos, todos sin embargo del mismo usuario.

III. Las cpu's de los datores son promiscuas. Esto quiere decir que los datores pueden ejecutar tanto los trabajos emanados del usuario "local", como los otros usuarios, importados (los trabajos) de otros datores a través de las facilidades de comunicación existentes. Generalmente, para que exista nivelación de carga, los sistemas distribuídos deben estar moderada o fuertemente acoplados, debido a la elevada ocupación de los canales de comunicación.

Compartición de datos en redes

Todas las redes actuales, incluyendo las débilmente acopladas, permiten el envío de datos hacia el programa que los requiere.

Si las redes son homogéneas y los datos a transmitirse son grandes, a menudo resulta más conveniente man-

El caso es el mismo que en el caso II anterior. Un usuario puede generar varios trabajos, y los datores poseen sistemas operativos de multiprogramación. Sin embargo, ahora un sistema operativo puede exportar carga hacia otro dator que se encuentre menos ocupado. Para compartir carga eficazmente es necesario poder compartir datos también, debido a que un programa (trabajo) que emigra de un dator a otro necesita llevarse consigo los datos a que hace referencia, parte de los cuales estarán en el dator del emigrante. El sistema distribuído que se está diseñando en el Departamento de Computación (IIMAS) de la UNAM poseerá entre sus características el de compartir cargas, es decir, el tener datores promiscuos.

dar el programa a donde están los datos. Esto no puede hacerse en redes heterogéneas.

Periféricos Prómiscuos. Los datores poseen cierta capacidad de almacenamiento periférico. Estos dispositivos pueden ser accesados por otros datores del sistema, por lo que la información que se posee se comparte entre todos los datores. Sin embargo, se entiende que la información local se accesa más rápidamente que la información foránea.

En sistemas fuertemente acoplados, los datores comparten memorias y dispositivos de almacenamiento masivo. En sistemas débilmente o moderadamente acoplados, a menudo los dispositivos periféricos pueden ser accesados directamente solo por el dator al que pertenecen, el que sin embargo tiene "la obligación" de leer de ellos datos y enviárselos a otros datores que los soliciten. De esta forma, cada dator "indirectamente" accesa todos los datos, asegurándose así su promiscuidad.

Al tener los datos descentralizados, en vez de tenerlos concentrados en la computadora central, se obtiene un mejor rendimiento de los sistemas de cómputo, ya que varios datos pueden estar procesándose separadamente (en paralelo, simultáneamente) en diferentes datores, y se evita asímismo el envío de datos locales a un lugar central de proceso. Esto último disminuye los costos de transmisión de datos. Se transmiten solo excepciones (regiomontano que fué a Mérida a cambiar su cheque), resúmenes (control de inventarios, control de gastos) o resultados.

Regla: poner la información dentro del dator que sirve al usuario que más usa esa información.

A veces se incluye esta otra regla: solo el dator local puede hacer actualizaciones a su información local, pero todos pueden consultarla. Viólase a veces, permitiéndose que varios datores hagan actualizaciones a una información foránea: actualizar el número de zapatos existentes en la bodega, desde varios puntos de venta.

Captura de datos/interacción hombre–máquina. La captación de datos se realiza en forma local, donde cada dator provee la inteligencia (en forma de programas de validación y verificación) para que los errores e inconsistencias se corrijan tan pronto como se cometen y detectan. Es decir, se pone un filtro oportuno a los datos erróneos, los que no llegan muy adentro del sistema de información.

Si se tienen datores promiscuos (con nivelación de cargas), entonces resulta adecuado tener un dator por cada usuario "consistente". En esta forma la verificación de errores de entrada es excelente. Y cuando el usuario se va y ya no ocupa su dator, la cpu de éste colabora a procesar trabajos de otros usuarios que han permanecido activos.

Como ejemplo de este sistema, en el laboratorio de Inteligencia Artificial del M.I.T. (Instituto Tecnológico de Massachusetts) han instalado un teclado y una pantalla por usuario, pero la memoria para el despliegue de caracteres y la capacidad de cómputo para tal despliegue de caracteres y la capacidad de cómputo para tal despliegue está centralizada en una pdp–11 satélite de la pdp–10, que es la máquina que procesa la información.

Si las pantallas son de alta resolución, del tipo de rasterscan-displays, y el sistema operativos es de multiprogramación, entonces es posible designar parte de la pantalla para la salida de un trabajo, otra parte para la salida de otro, etc. En esta forma es posible que el mismo usuario esté trabajando en el mismo dator con dos o más programas suyos, interactivos y concurrentes. Tal sucede con las máquinas PERQ. Y si existe nivelación de carga, pudiera suceder que sus diferentes trabajos estuvieran siendo procesados en datores distintos.

La idea de que los datores desocupados contribuyan al desfogue de trabajos que les corresponden a datores "ocupados" es buena, puesto que permite un uso más eficiente de la capacidad de cómputo instalada. Si yo soy el usuario o dueño de un dator, y no lo estoy usando, qué bueno que pueden usar de él otros usuarios más trabajadores que yo. Esto no ocurre con los teléfonos, por ejemplo: los aparatos telefónicos de las oficinas que están cerradas no los puede usar nadie. Sin embargo, esto sí ocurre con los selectores para cada teléfono: los teléfonos desocupados no usan selectores, y éstos quedan libres para su uso por los teléfonos ocupados. (Los selectores están centralizados en la central telefónica). Lo que aquí se propone es el uso promiscuo de recursos distribuídos.

Acceso a información remota.

Si imaginamos un sistema de datores donde cada uno posee parte de la información total o base de datos, entonces hay dos formas substancialmente diferentes de usar la información:

1. cada dator usa su información local o cualquier información foránea, sin mezclarlas o mezclándolas poco y de manera simple.

Nivelación estática de carga

En un sistema de bases de datos distribuidas, la carga total puede nivelarse preasignando funciones y segmentos de la base de datos a los datores, en forma estática. Este método es usado a menudo en un sistema distribuído débilmente acoplado, el que puede estar formado por computadoras heterogéneas, pequeñas, medianas y microcomputadoras. Las funciones que se deben proveer son: (1) asegurar acceso remoto y local a las partes de la base de datos distribuída, y (2) trasladar las diferentes partes de la base de datos de uno a otro dator.

Por ejemplo, un dator puede contestar una pregunta mediante un acceso a su base de datos local, o a la base de datos foránea. O bien la pregunta puede responderse con 95 por ciento de la información local y 5 por ciento de la foránea, o viceversa.

2. Para contestar una pregunta o resolver un problema, el dator debe accesar varias bases de datos y conjuntar, mezclar y entreverar las informaciones parciales obtenidas, de una manera compleja y por un número sustancial de veces.

Nivelación dinámica de carga

Un sistema distribuido puede usarse para desparramar dinámicamente la carga de procesamiento total, y la carga de acceso a la base de datos, a través de los datores disponibles.

Si todos los datores poseen la misma capacidad, los trabajos o transacciones pueden moverse libremente entre ellos. Sin embargo, puede ser deseable ejecutar un trabajo específico en una computadora particular, porque requiera acceso a una base de datos asociada a esa máquina. En otras ocasiones puede ser deseable mover trabajos, tal vez acompañados por sus datos, de un dator sobrecargado a otro que posee capacidad disponible.

Para esto se necesita un desdoblador de preguntas, capaz de transformar una pregunta dirigida a toda la base total en una función lógica de preguntas parciales, las que al combinarse por un integrador de respuestas nos brinda la respuesta final a la pregunta inicial. Por ejemplo, la pregunta "¿cuántos niños varones entre 6 y 10 años tuvieron tosferina el año pasado en toda la República? "

puede desdoblarse, suponiendo que existe un dator por cada estado, en 30 preguntas, cada una por cada estado e, del siguiente tipo: "En tu estado e, y para la edad y variando de 6 a 10, dime cuántos niños hubo con tosferina con edad j el año pasado". La integración de las respuestas es simple, y consiste en sumar los arrojos parciales.

El desdoblamiento de preguntas requiera (si se realiza en forma automática) de conocimientos profundos, por parte del desdoblador, de la constitución de las bases de datos parciales; se necesita introducir variables, como el estado y la edad en el ejemplo anterior, sobre las cuales cuantificar "para todas las edades j y todos los estados e, . . . ".

Acceso a procesamiento remoto. La nivelación de carga de la que ya se habló puede realizarse desde una manera muy sencilla hasta una forma muy compleja, como se esboza a continuación.

a. Sólo un dator para un trabajo. Esta disciplina la siguen la mayor parte de las redes de procesamiento remoto. Una terminal en un extremo de la red usa una computadora en el otro extremo. La terminal (el proceso) ignora y no se beneficia de la gran cantidad de recursos que están también disponibles. Un trabajo entra completamente a un procesador (éste, sin embargo, estará por lo general ejecutando varios trabajos concurrentemente) a ejecutarse.

b. Paralelismo entre trabajos. Cuando un usuario genera varios trabajos, éstos pueden emigrar hacia diversos datores

distintos. Cada dator posee su sistema operativo de multiprogramación, y ejecuta sus diversas tareas concurrentemente, o usando una disciplina de tiempo real.

Este esquema se usará en el Sistema de Procesamiento Distribuido que se está diseñando en el Departamento de computación (IIMAS) de la UNAM.

c. Paralelismo dentro de un mismo trabajo. Un solo trabajo (task) se subdivide en partes más pequeñas, las que se "exportan" a otros datores para ser procesados por ellos, y cuyas respuestas se integran para constituir el resultado final. Este tipo de paralelismo es el más difícil de lograr, pues se requiere de la capacidad de segmentar automáticamente un trabajo en varios subtrabajos que se puedan después reintegrar (sus resultados) fácilmente.

En la máquina AHR que está siendo construida en el Departamento de Computación (IIMAS) de la UNAM se logra un paralelismo dentro de un mismo trabajo, ya que varias docenas de microcomputadoras procesan y evalúan un mismo programa escrito en LISP (un lenguaje de alto nivel).

Control del sistema distribuido de datores. Las labores de control y supervisión de una red o sistema distribuido de datores son, entre otras:

- nivelación de carga
- estadísticas de funcionamiento
- reconfiguración
- detección y aislamiento de datores y líneas defectuosas.

El control global (y cada una de sus funciones en particular) puede llevarse a cabo en una de las siguientes formas:

1) no existente.
2) Control centralizado. Cierto dator específico ejerce el control.
3) Control descentralizado. Cada dator efectúa ciertas labores de control, las que están diseñadas de tal suerte que se logra el efecto de control, sincronización, reconfiguración, etc., deseado.

TRES TIPOS ESPECIFICOS DE SISTEMAS DISTRIBUIDOS

Para finalizar el artículo, se presentan tres máquinas de procesamiento distribuido que están siendo diseñadas y construidas en el Departamento de Computación (IIMAS) de la UNAM. Cabe mencionar que han surgido de ideas originales, de diseños propios, no copiados de universidades o instituciones extranjeras.

Máquinas heterárquicas: Proyecto AHR. Esta arquitectura de cómputo consiste de varias docenas de microprocesadores, fuertemente conectados, que les permite ejecutar en paralelo un solo programa. El programa está escrito en lisp, por lo que es válido decir que la máquina AHR procesa LISP como lenguaje de máquina. Los datores comparten carga, la que es ejecutada en cada dator a nivel de un nodo. Básicamente un modo es una instrucción primitiva de LISP. La arquitectura de hardware de la máquina es tal que solo los nodos que pueden ser ejecutados en un momento dado son enviados a los datores para su ejecución. Todos los datores tienen acceso común a la memoria de variables (donde residen los diferentes valores de las variables de LISP), a la memoria pasiva (donde yacen los datos y los programas que no están activos) y a la parrilla (donde los programas activos están siendo evaluados o procesados).

Los datores no poseen memoria masiva ni teclados. Hay una conexión con una minicomputadora (llamada "caja inteligente") que es la que atiende a los usuarios y posee periféricos. Es decir, la máquina AHR puede considerarse como un periférico de esta minicomputador. . Este "periférico" tiene la habilidad de transformar programas (en LISP) en resultados.

Datores en un cable coaxial. El Proyecto de Sistemas Distribuidos (IIMAS) de la UNAM contempla la conexión a alta velocidad de varios datores homogéneos, cada uno de los cuales posee memoria masiva, teclado y pantalla. Se prevee que cada sistema operativo sea de multiprogramación. Se planea en la nivelación de cargas mediante la exportación de trabajos de los datores más ocupados a los menos atareados. Es decir, la arquitectura tendrá datores promiscuos, así como también periféricos promiscuos. El sistema es fuertemente acoplado. Se usa paralelismo entre trabajos, no dentro de cada trabajo (como lo hace la máquina AHR). Probablemente exista un control descentralizado de el sistema de datores.

Este proyecto, incipiente aún, promete ser una paradigma para el diseño y evaluación de diferentes ideas en el desarrollo de sistemas distribuidos comerciales y operacionales.

Máquina de bases de datos. Este proyecto contempla la construcción de un dator especializado, capaz de manejar eficientemente una base de datos mediana o grande. Se han analizado las operaciones principales que se efectúan en el manejo cotidiano de un sistema de información, y se ha propuesto la construcción de hardware especial (mergers, sorters, buscadores, memorias asociativas, direccionadores por contenido, memorias de caché, memorias de burbujas, dispositivos de cargas acopladas, etc.) y la modificación de equipos electrónicos ya existentes, para adecuarlos a las necesidades de la máquina de bases de datos.

Probablemente, la máquina de bases de datos se conectará como un periférico de un dator general, y funcionará como un procesador de "back-end", en tanto que la atención a los usuarios la realiza el dator general, ayudado y en coordinación con el sistema total de datores. Es decir, la máquina de bases de datos se conectará al resto del sistema de datores en forma análoga a la conexión de la máquina AHR a su "caja inteligente".

CONCLUSIONES

Los sistemas de procesamiento distribuido ofrecen reducciones sustanciales en costo, así como ventajas en versatilidad, confiabilidad, crecimiento incremental, y disponibilidad, por lo que se espera que su uso vaya en aumento constante y desplace a los sistemas tradicionales basados en una computadora central que comparte su tiempo entre usuarios que emplean terminales clásicas.

AGRADECIMIENTOS

El autor agradece la invitación del Dr. Héctor Nava Jaimes, Jefe del Departamento de Ingeniería Eléctrica del Centro de Investigación y Estudios Avanzados del Instituto Politécnico Nacional, a participar temporalmente como Profesor Visitante en su Departamento. Este trabajo se desarrolló en el CIEA del IPN, después de fructíferas discusiones con el Dr. Juan Milton Garduño, Profesor de Telecomunicaciones del mencionado Departamento, y con otros colaboradores del CIEA.

Los proyectos "Máquina AHR", "Computación Distribuida" y "Máquina para bases de datos" fueron originados por el autor en el Departamento de Computación del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (UNAM), y deben su existencia a la asiduidad y constancia de sus integrantes, y al apoyo del Dr. Tomás Garza, Director del Instituto. El autor reconoce la valiosa ayuda prestada por los integrantes de estos proyectos (Luis Lyons, Luis Hugo Peñarrieta, Dora Gómez, Renato Barrera, David Rosenblueth, Kemer Norkin, Angel Kuri, entre otros), vertida a través de numerosas discusiones e innumerables tazas de café.

El intercambio de correspondencia con el Prof. Carl Hewitt, del Laboratory for Computer Science del Massachusetts Institute of Technology, quien ha popularizado el concepto de "actores", ha sido también influyente en las ideas de este artículo.

El trabajo aquí descrito ha sido patrocinado parcialmente por el Consejo Nacional de Ciencia y Tecnología (Proyecto 1632), México.

BIBLIOGRAFIA

1. Tutorial on Distributed Processing. Second Edition. Burt H. Liebowitz, John H. Carson. IEEE Computer Society. 5855 Naples Plaza. Long Beach, Cal. Publicación No. EHO 127-1. 1978.

2. Jon Roland. Microtecnología para las masas. Ciencia y Desarrollo 28, Sept-Oct. 1979, 125-130.

3. Carl Hewitt. Design of the APIARY for knowledge-based systems. Working papel 186 B, July 1978. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Mass.

4. Velarde, C. y Rosenblueth, D. La máquina AHR para procesamiento en paralelo, etapa 1. Comunicación Técnica AHR-79-2. IIMAS-UNAM. 1979.

5. Guzmán, A. and Segovia, R. A parallel configurable LISP machine. Proceedings of the International Conference on Systems Sciences. Patras, Greece. 1976. Also available as: Comunicaciones Técnicas IIMAS No. 133, Vol. 7. 1976. IIMAS-UNAM. (Technical report AHR-76-1).

centro de educación continua
división de estudios de posgrado
facultad de ingeniería unam

CASOS PRACTICOS DEL DISEÑO DE SISTEMAS DE INFORMACION

DIGITAL MODEL FOR THREE-DIMENSIONAL SURFACE REPRESENTATION

DORA GOMEZ

ADOLFO GUZMAN ARENAS

AGOSTO, 1980

Geo-Processing, 1 (1979) 53–70

DIGITAL MODEL FOR THREE-DIMENSIONAL SURFACE REPRESENTATION

IXIA GOMEZ and ADOLFO GUZMAN

Computer Science Department, IIMAS, National University of Mexico, Mexico 20 D.F. (Mexico)

ABSTRACT

Gómez, D. and Guzmán, A., 1979. Digital model for three-dimensional representation. Geo-Processing, 1:53–70

A tree of planar or spherical triangles is used to represent a 3-d surface; rather flat regions will be represented by large triangles, while abrupt zones will require further subdivision of the model into smaller triangles. Their vertices are not placed on a regular grid; they are allowed to fall at (or near) places such as ridges and peaks, where the change in slope is significant.

Starting from a collection, not necessarily good or complete, of "significant" points, the model selects five of them to form four triangles. Each triangle either matches the surface within a prespecified error tolerance, or else is further subdivided, by selecting appropriate "significant" points, into four triangular sons, which then receive in turn the same treatment. The tree stops growing when all the surface is represented within the specified tolerance. The model consists of the vertex points arranged into a table suitable for quick retrieval and interpolation.

Thus, the model guides its own construction; its components (points) are taken from the set of "significant" points, not in an arbitrary fashion but only where and when needed. Since the model proposes the approximate location of the next point to be included in it, the set of "significant" points may be small or non-existant.

A constant signal to noise ratio and a representation thrifty in storage are achieved in this manner.

The model is being tested for use in digital representation of terrain elevation. Large savings in memory are expected, when compared to contour lines storage, for instance.

The paper concludes with some comments in favor of the use of this model to describe gray level pictures.

INTRODUCTION

The digital representation of three-dimensional surfaces plays important roles in photogrammetry and cartography (drainage patterns, contour lines (Gómez, 1978), valleys formation, stereoscopy (Gómez), dimensions of the human body); scene analysis (occlusion of bodies, explanation of regions (Guzmán, 1971) and objects, range finding, shape from shading (Horn, 1970); computer graphics (hidden lines, shadows, coloring, specular reflexions); image processing; remote sensing (thickness of ice (Jensen, 1976), underground geology), and other disciplines.

The surfaces to be considered are of the following types:

$$z = f(x,y) \qquad (I)$$

$$g(x,y,z) = 0 \qquad (II)$$

Surfaces of type (I) give a single z value ("height") at each pair of coordinates x,y; surfaces of type (II) can represent more general surfaces in the space, for instance the skin of a hand (Figure 1). The examples of the paper refer only to surfaces of type (I), but the model hereby proposed can be used for both types.

Our model for a surface consists of a minimum complete cover of triangles; that is, a mutually exclusive, collectively exhaustive finite set of triangles such that (1) each point in the surface is represented by exactly one triangle[1] and (2) every triangle represents at least one point of the surface. To this we add the important restriction: (3) the difference between the coordinates x,y,z of a point in the surface and those x',y',z' of its representative (as given or computed from the model) is less than a pre-specified error or tolerance ε. Thus, for two tolerances $\varepsilon_1 > \varepsilon_2$, the same surface will be represented by two models M_1 and M_2, where M_2 is a refinement of M_1.

For surfaces of type (I), we make x'=x, y'=y and the distance between the real point and the model point is just the difference in heights.

Non uniqueness of the model

Several complete covers for a surface exist; in order to save memory, it is preferable to use a cover with fewer triangles (hence, each triangle covers in the average a larger area), as long as they obey restriction (3) above. The idea is: (4) not to subdivide a triangle unless it fails to represent reality within the ε tolerance.

Even so, a 3-d surface can be represented by several models complying with (4) above. There is no unique model; by changing the position of the original rectangular frame a new representation is obtained. A different starting rectangle in the procedure or a different orientation of the grid will produce a different cover of triangles.

---

[1] The width of the sidewalk is a constant fraction k_1 (say, 10%) of the corresponding median: $w_{SW} = k_1 \cdot s-r$. To incorporate these sidewalks to the model, it was only necessary to modify the definition of the function *inside*, so that a point is "inside" triangle A d a if it falls inside it or at the sidewalk (Refer to fig. 7). If not significant point r is found near enough r'' so as to meet the 10% requirement, the model orders to find (to fabricate) a new significant point with coordinates x,y closer to r_x'', r_y''; the model only accepts significant points close enough to r'' (r'' is the midpoint of $(c_x,c_y,0)+(d_x,d_y,0)$) to ensure that the resultant flaps will not be wider than k_1. An optimal way (suggestion 1) is needed to compute k_1.
The sidewalks slightly contradict the assumption (1), uniqueness of representation for a point, of the introduction. This is of no importance.

This is not a problem for 3-d surface representation, as used for instance in applications to cartography and computer graphics. For surface comparison it is

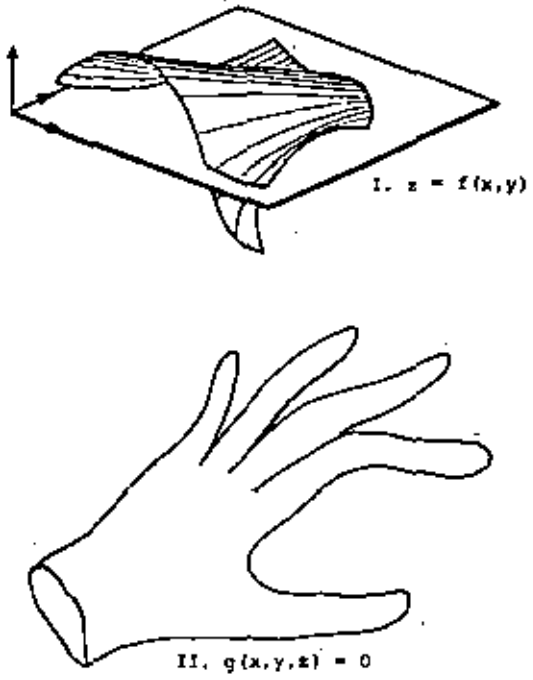


Fig. 1. TYPES OF SURFACES. The model described in this paper is able to represent either (I) single height surfaces, or (II) more general surfaces.

much better to have a unique (canonical) model, perhaps through a normalization procedure.

The set of significant points. Using a method external to the model, for instance stereoscopy (Gómez), gradient extraction (Signor and Nadler,1978), river following, or others, an initial set of "significant" points is chosen on the 3-d surface that we want to represent. A point is called "significant" if in its neighborhood the change in slope is large.

The model begins by using some of these points; if it later finds necessary to grow, it indicates the approximate place (x,y coordinates) where a new "significant" point should be added to the model.

The model thus consists of a subset of "significant" points, defining a triangular irregular mesh; if the original set of "significant" points is too small, the model will suggest where to add one; if too many, most of them will be ignored (not included in the model); if the procedure that implements "significance" is noisy or unreliable, the model still guarantees the ε tolerance, but storage economy suffers.

Therefore, in a computer implementation, it is not necessary to obtain first the set of significant points and then to pick the model from them; instead, the

model can begin to grow as soon as five or six are found, and the procedure that extracts significant points is called by the model as it deems necessary.

Obtention of the three-dimensional surface.

It is assumed that the surface to be modelled already was obtained and exists available in some suitable representation, v.gr., a 2-d matrix containing height values. This data could have been obtained by stereocorrelation (Gómez) of a pair of pintures, by interpolation of digitized contour lines (Bribiesca and Avilés, 1974) or by other means.

CONSTRUCTION OF THE MODEL

In order to describe the model, it is necessary to explain

(1) its constituent parts. In this case, they are vertices ("significant" points from the 3-d surface to be represented) that form planar, but tilted, triangles.

(2) how the model is sotred: the data structure used to keep the model in memory (primary or secondary storage). A tree of triangles, each with none or four sons, is used.

(3) the use of the model: the procedure to follow for reconstruction of the 3-d surface from the model; the way to obtain the coordinates of a point in the surface from the cover of triangles. Here a directed access is used to the correct triangle starting from the top of the tree of (2), and falling down the appropriate chain of triangle sons, using little search and no backtracking.

(4) the construction of the model, i.e., the obtention of its parts from the 3-d surface. A recursive procedure will be presented, where the model guides its own construction, by suggesting places (x,y coordinates) where to incorporate into itself points from the 3-d surface that are also " significant" with respect to changes in slope.

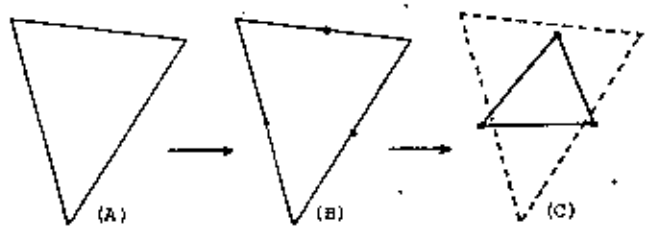


Fig. 2. TRIANGLE REFINEMENT. If it is necessary to refine triangle (A), three new vertices are proposed at the mid-points (B) of the sides; "significant" points are located near those mid-points; once they are found (C), four new triangles stand instead of the original (A).

The parts of the model

To represent a surface $z = f(x,y)$, the model uses a collection of planar tilted triangles; each of them is defined by its three vertices, chosen to lie on the surface $z = f(x,y)$ to be represented.

All the points inside the triangle are interpolated linearly: the surface inside the triangle is considered flat (but not horizontal, in general). Since the real surface $z = f(x,y)$ is not flat, an error is introduced by this assumption. If everywhere in the triangle this error (height difference) does not exceed a tolerance ε, the planar triangle is considered to be a good (and final or "terminal") representative for that region of the surface, and it is included in the model. If the error is larger, the triangle is discarded by dividing it into four smaller triangles, each of which in turn undergoes the same treatment.

Initially the surface is divided into a small set of arbitrarily chosen large triangles; if the surface is bound by a rectangle (as it is frequently the case in maps), four triangles are chosen as shown in part C of Figure 3.

The final model contains triangles (of different sizes) that represent the surface $z = f(x,y)$ with a tolerance ε. Each of the vertices of these triangles was proposed by the model by dividing a triangle in four through inclusion of new vertices near the middle points of the sides (Figure 2).

Once every triangle is refined, the vertices (E in Fig. 3) are stored in an appropriate way, suitable for quick data retrieval for surface reconstruction.

When to stop refining

A triangle such as in Figure 2A is refined further, unless

1) the difference between the real height $z = f(\bar{x},\bar{y})$ and the computed height $\bar{z}$ at the center of mass $(\bar{x},\bar{y},\bar{z})$ of the triangle is smaller than ε, and
2) every point in a grid of points spaced at most K units apart and inside A is within ε of the real point on the surface $z = f(x,y)$.

Test (1) is a quick test; test (2) is applied only if (1) does not find a difference exceeding ε.

K, the distance between two points in the grid of (2), is a function of ε; normally, $K = \min(\varepsilon/m, K_0)$, where m is the mean slope of the surface at the triangle (A), and K_0 is the diameter of the smallest topographic feature (hill, ravine) that it is necessary to represent in the model. Generally K_0 is given by the user of the model; "be sure to check the model every 500 horizontal meters for accuracy"; then $K_0 = 500$.

Flow diagram. The procedure for construction of the model could be summarized as:

- Let T be the set of triangles that are candidates to be included in the model.

  Initialize T with the four triangles of (C), Fig. 3.

- Mark every triangle of T as "terminal" if it passes tests (1) and (2) of the Section "When to stop refining". If these tests fail for a triangle, mark it "non-terminal", divide it into four sons (cf. Fig. 2) and add them to T.
- Exit when all triangles of T (including all the additions to T) are marked (either "terminal" or "non-terminal"). Then T is the model.

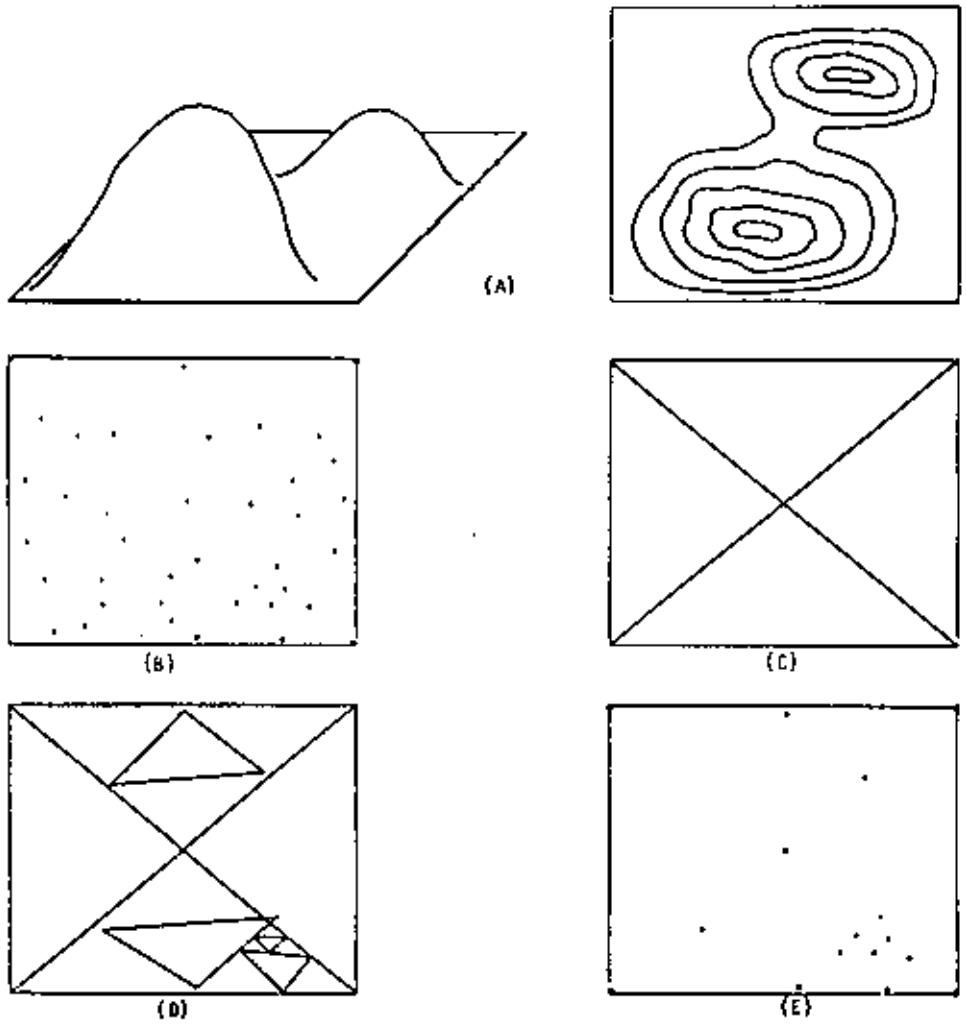


Fig. 3. MODEL BUILDING. (a) the surface, (b) the "significant" points, (c) the four initial triangles, (d) the final triangles, (e) the points of the model, usually a subset of (b). Triangles (d) are in the space (they project out of the paper); similarly, points (e) have three coordinates.

TABLE I. FLOW DIAGRAM FOR MODEL BUILDING.

This simple program constructs surface models such as that shown in fig. 6.

```
BEGIN
    T  the four initial triangles of fig. 3C;
    for every triangle in T
        if it passes tests (1) and (2) of section "When to stop refining"
        then mark it 'terminal'
        else mark it non terminal   and
             add its four sons to T;
END.
```

A non-terminal triangle is not needed in the model, since

(1) its accuracy is worse than ε, and

(2) some of its descendants are a *finitive* terminal triangles, hence suitable for modelling.

Thus, the model could be just the collection of terminal triangles.

This is advisable when the cover is made of similar triangles (q.v.), where it is easy to pick up the correct triangle for surface reconstruction. If the triangles are not similar, it is preferable to retain the non-terminal triangles into the model. This facilitates the addressing of the correct terminal triangle that gives the height Z of a point (x,y) (i.e., the point (x,y,Z) that represents the point (x,y,z) of the 3-d surface). More of this in the section IV 'Data Retrieval for Surface Reconstruction'.

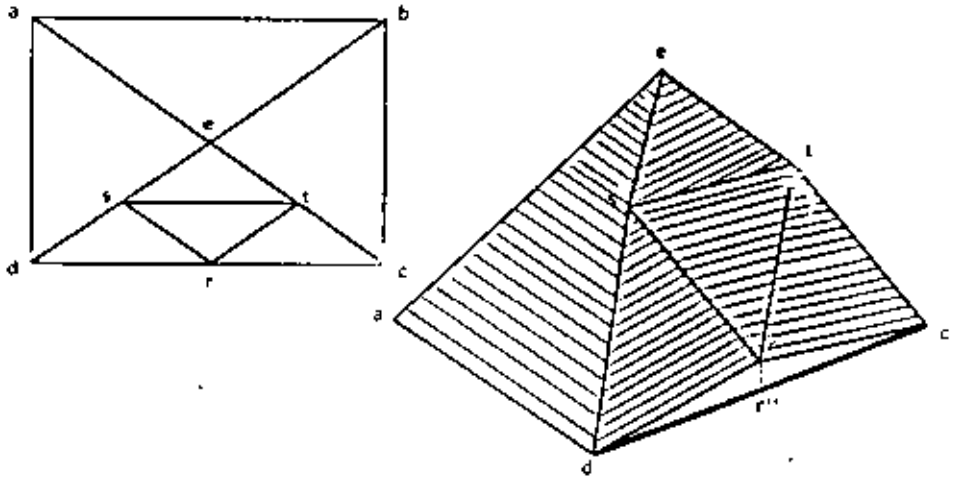


Fig. 4. MODEL VISUALIZATION. We try to give an isometric view of the appearance of the triangular model in 3-d space. Point $S=(s_x,s_y,s_z)$ does not lie on the 3-d line d - u; but point $(s_x,s_y,0)$ does lie on the line $(d_x,d_y,0)$ - $(e_x,e_y,0)$. Triangles such as d e s or d t t are not part of the model: they represent no part of the real 3-d surface because they lie vertical. They are useless.

Cover of similar triangles

Two polygons are similar if the corresponding angles are equal, the sides parallel and their length proportional.

If in Figure 2 we stop the refinement at (B), choosing the midpoints as new vertices to include in the model, the final cover of the model is composed of two families of similar triangles, because a line joining the middle points of two sides is parallel to the third side.

A word of caution: the triangles are not similar as they lie in the 3-d space. Their projections on the plane x,y do form a family of similar 2-d triangles (for triangles *a b e* and *e d e* of Fig. 4, and all their descendants) another family of similar 2-d triangles for triangles *d a e* and *b e e*, and all their descendants.

The advantages of the cover of similar triangles are:

(a) storage of these triangles is easy. (Klinger and Nikitas) stores a hierarchy of squares.

(b) reconstruction of the surface from the model becomes simplified.

(c) a set of "significant" points (B in Fig. 3) is not needed.

The disadvantage comes from (c):

(d) the model might contain more points, since they are not special or significant: they are not the best to choose for interpolation of planes.

DATA STRUCTURE FOR MODEL STORAGE

This section describes the way to organize the storage of the model. Essentially, the storage consists of a collection of triangles. Each triangle is stored in a "frame"; each of them contains

- three internal vertices
- a "terminal" or non-terminal mark for each son.

The terminal mark (zero) indicates that a triangle son already fulfills the ε accuracy, hence it (the son) has no sons of its own --need not be further subdivided.-- The non-terminal mark, an integer different from zero, indicates the location (frame) in the model matrix occupied by this triangle son. Thus, when a node is marked as non-terminal, the mark itself also says where (in what frame) that son is stored. See Fig. 5 and Table II. Slightly different conventions were used in IIMAS-UNAM (Gómez, 1978).

The model is stored in a matrix (C, Fig. 5) which is a collection of frames. A non-terminal triangle occupies a frame; it stores clockwisely (B, Fig. 5) its three central vertices and a mark specifying for each son whether it is terminal or not. A terminal triangle does not use a frame, since it has no sons. But a non-terminal triangle could very well have four terminal sons. That is the case of frames 4 to 9 of Fig. 6.

The initial frame, frame 1, is stored in a slightly different manner (part A of Fig. 5, because it describes a rectangle.

A more complicated example is given in Fig. 6.

Storage of vertices. When describing a non-terminal triangle (e.gr., triangle 1 2 3 of b, Fig. 5, only vertices 4, 5 and 6 are stored in the frame belonging to that triangle 1 2 3, since vertices 1, 2 and 3 were undoubtely stored in the ancestors of triangle 1 2 3. This avoids multiple storage of vertices, and exploits the fact that in order to examine whether a point (x,y) fallsinside the

TABLE II. NAMING CONVENTIONS.

These conventions are important for correct storage of vertices (such as 4 of the internal triangle P), and its subsequent appropriate retrieval for reconstruction of the 3-d surface. For a use, see definition of procedure 'altitude' in section "Data retrieval for surface reconstruction".

CONVENTIONS I. Refer to part (A) of Fig. 5.

Vertices of triangles which are sons of the rectangle are named as shown. The correct names for (A) are:

rectangle: a b c d
M = triangle a b e
N = triangle b c e
O = triangle c d e
P = triangle d a e

CONVENTIONS II. Refer to part (B) of Fig. 5.

Vertices of triangles that are sons of triangles are named clockwisely, starting with the vertex that also belongs to the father.

If the triangle to be named is the internal triangle (P), then start with the vertex that falls near the middle point of line 1 → 2, where 1 is the first of the vertices that belong to the father, and 2 is the second of them.

The correct names for triangles of (B) are:

triangle 1 2 3 (first vertex is 1)
M = triangle 1 4 6
N = triangle 2 5 4
O = triangle 3 6 5
P = triangle 4 5 6

2-3[2] triangle 1 ? 3 or not, we already asked a similar question to the ancestors of 1 ? 3. In this way the coordinates fo vertices 1, 2 and 3 are already known when triangle 1 ? 3 is accessed (cf. Section 'Data Retrieval for Surface Reconstruction').

A vertex is stored by storing its three coordinates x, y, z. Some duplication (not triplication or multiplication) occurs when a vertex such as 5, 20 or 15 in Fig. 6 gets stored by two non-terminal brother triangles. For instance, vertex 5 is stored at frame 4 that describes triangle 3 1 9, and also at frame 3 that

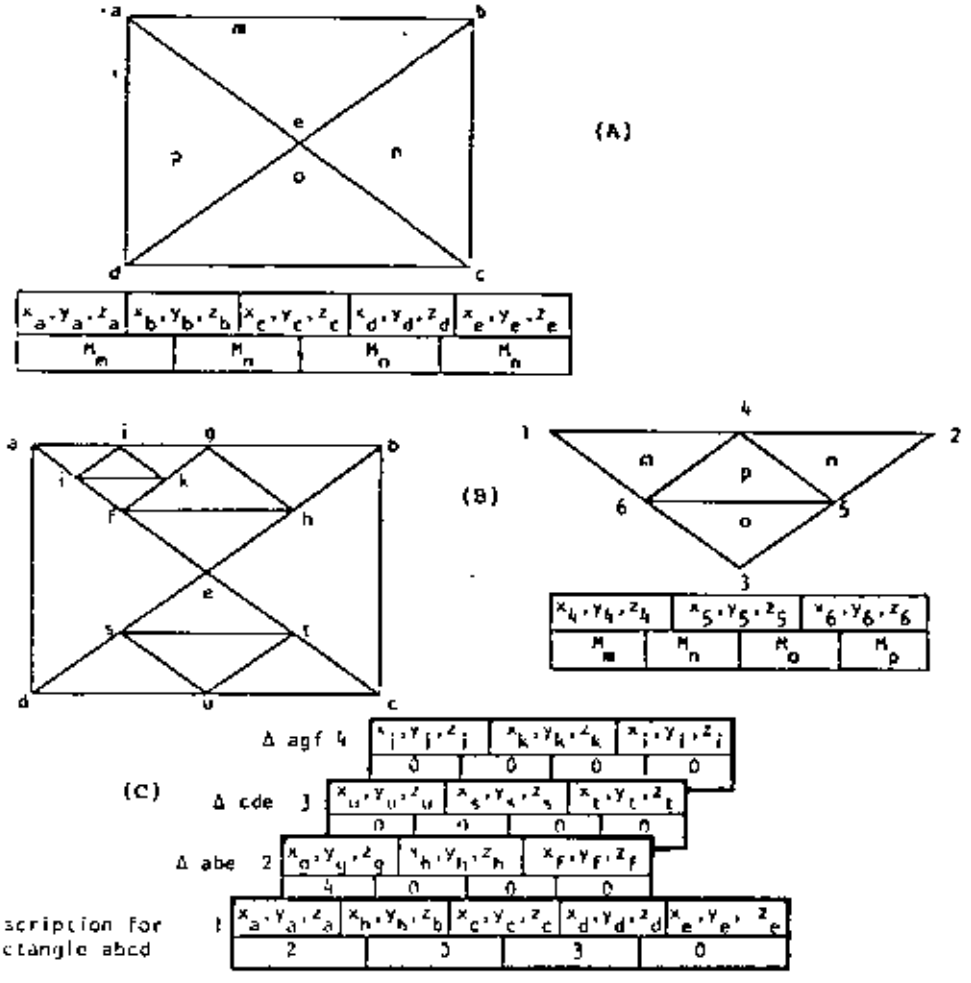


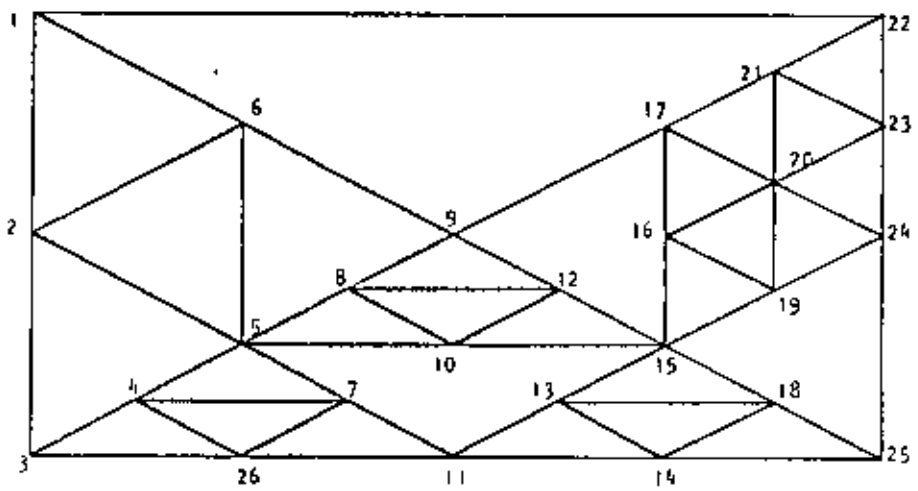
Fig. 5. DATA STRUCTURE. (A) Storage conventions for the initial rectangle. (B) Storage conventions for a non-terminal triangle 1 ? 3. Its sons are M, N, O, P. (C) Example of a model and its data structure. Only a non-terminal triangle uses up a frame. The 2 0 3 0 marks of frame 1 mean that son M is non-terminal and it is described in frame 2, son N is terminal, son O is non-terminal and it is described in frame 3 and son P is terminal (mark = 0 means terminal). The model is stored in a matrix (C) which is a collection of frames.

describes triangle 2 5 3 9. The trivial cure will be to keep a table of vertices, and to store in the frame pointers to the table, instead of the three coordinates x,y,z.

This table of vertices is not used in our model because it saves little storage:

(1) If both a pointer and a vertex coordinate occupy a word of memory, then to use the table requires 2 pointers + 3 coordinates = 5 words; not to use the table requires 3 coordinates + the same 3 coordinates = 6 words;

(2) if for some reason triangle 25 3 9 selects vertex 5 as the "significant" point near the mid-point of side 3-9 (Refer to Fig. 5), but triangle 3 1 9 selects vertex 5' (a different vertex, near vertex 5 but not the same) as the "significant" point near the mid-point of side 9-3, then the table wastes memory.



	Frame	VERTEX a	b	c	d	e	Δ M	Δ N	Δ O	Δ P
[□ 1 22 25 3]	1	1	22	25	3	9	0	2	3	4

		VERTEX 4	5	6				
[Δ 22 25 9]	2	24	15	17	5	0	0	6
[Δ 25 3 9]	3	11	5	15	7	8	9	0
[Δ 3 1 9]	4	2	8	5	0	0	0	0
[Δ 22 24 17]	5	23	20	21	0	0	0	0
[Δ 24 15 17]	6	19	16	20	0	0	0	0
[Δ 25 11 15]	7	11	13	18	0	0	0	0
[Δ 3 5 11]	8	26	4	7	0	0	0	0
[Δ 9 15 5]	9	5	12	10	0	0	0	0

Fig. 6. MODEL EXAMPLE. This example was constructed using the rules (A) and (B) of Fig. 5 and Table II.
Each frame consists of vertices and pointers to other frames. Only non-terminal triangles occupy a frame of the matrix. This matrix is the model.

Simplified storage for cover of similar triangles

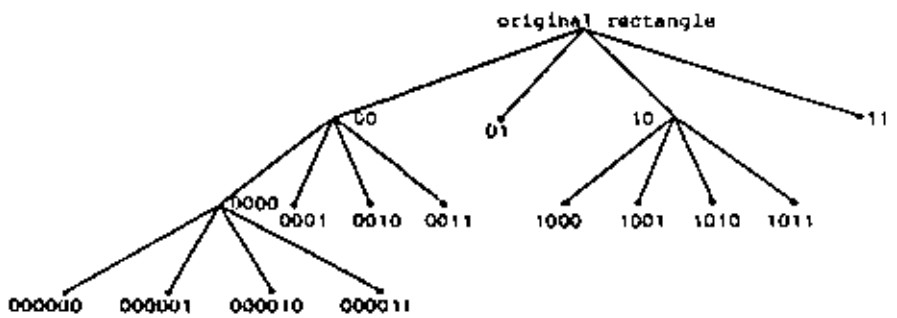
If we assume that the rectangle a b c d (Fig. 5) is a square and that the "significant" points are exactly at the mid-points of the sides of the triangles, instead of near them, then all the two-dimensional triangles[2] are similar (in fact, they are isosceles right angled triangles) and the (x,y) coordinates of any vertex need not be stored, since they are the average of the (x,y) coordinates of the vertices of an appropriate side.

The new representation for square a b c d of Fig. 5 is:

	Frame #	vertex a	vertex b	vertex c	vertex d	vertex e	Δ Δ Δ Δ M N O P
(rectangle a b c d)	1	Z_a	Z_b	Z_c	Z_d	Z_e	2 0 3 0
			vertex 4	vertex 5	vertex 6		
(triangle a b e)	2		Z_g	Z_h	Z_f		1 0 0 0
(triangle c d e)	3		Z_i	Z_j	Z_k		0 0 0 0
(triangle a g f)	4		Z_l	Z_m	Z_n		0 0 0 0

If the original area is not an square but a rectangle, we will have two families of similar two-dimensional triangles.

If we denote the sons M, N, O and P by 00, 01, 10 and 11, then we could turn from Figure 5 the following tree:



First generation	Second generation	Third generation
Δ a b e = 00	Δ a g f = 00 00	Δ a f i = 00 00 00
	Δ b h g = 00 01	Δ g h j = 00 00 01
	Δ e f h = 00 10	Δ f i h = 00 00 10
Δ b c e = 01	Δ g h f = 00 11	Δ j i h = 00 00 11
Δ c d e = 10	Δ e l m = 10 00	
	Δ d m n = 10 01	
	Δ e l m = 10 10	
	Δ m n l = 10 11	
Δ d a e = 11		

These codes could be combined with the z values to render a compact model. We do not pursue this further. In a similar manner, a tree of squares can be represented (Klinger and Nikitas).

DATA RETRIEVAL FOR SURFACE RECONSTRUCTION

In order to recover the 3-d surface, it is sufficient to ask the model what is the z value for any pair x,y. This is realized by the function ALTITUDE.

```
ALTITUDE (x,y)      % returns the height z of the point (x,y) as obtained from
                    % the model.  It is defined as:

a := MODEL |1,1|;   % first vertex of frame 1.  Frame 1 is the rectangle.
b := MODEL |2,1|;   % MODEL |*,L| is the frame L, a non-terminal triangle.
c := MODEL |3,1|;   % MODEL |*,*| is the matrix containing the whole model.
d := MODEL |4,1|;
e := MODEL |5,1|;
m := MODEL |6,1|;    n := MODEL |7,1|;    o := MODEL |8,1|;
p := MODEL |9,1|;   % retrieving the pointers to the sons.
error := -1;

ALTITUDE := if inside (a,b,e,x,y)
            then      if m=0 then height (a,b,e,x,y)
                      else ZETA (a,b,e,x,y,m)
            else if inside (b,c,e,x,y)
            then      if n=0 then height (b,c,e,x,y)
                      else ZETA (b,c,e,x,y,n)
            else if inside (c,d,e,x,y)
            then      if o=0 then height (c,d,e,x,y)
                      else ZETA (c,d,e,x,y,o)
            else if inside (d,a,e,x,y)
            then      if p=0 then height (d,a,e,x,y)
                      else ZETA (d,a,e,x,y,p)
            else error;
END ALTITUDE.
```

Function INSIDE (a,b,c,x,y) is true if the point (x,y,0) is inside the triangle $(a_x, a_y, 0)$, $(b_x, b_y, 0)$, $(c_x, c_y, 0)$ with sidewalks (see Fig. 7).

A point p is inside triangle a b c if p and c fall on the same side of a b and p and b lie on the same side of a c, and p and a rest on the same side of b c. A thesis (GÖmez) contains listings and results.

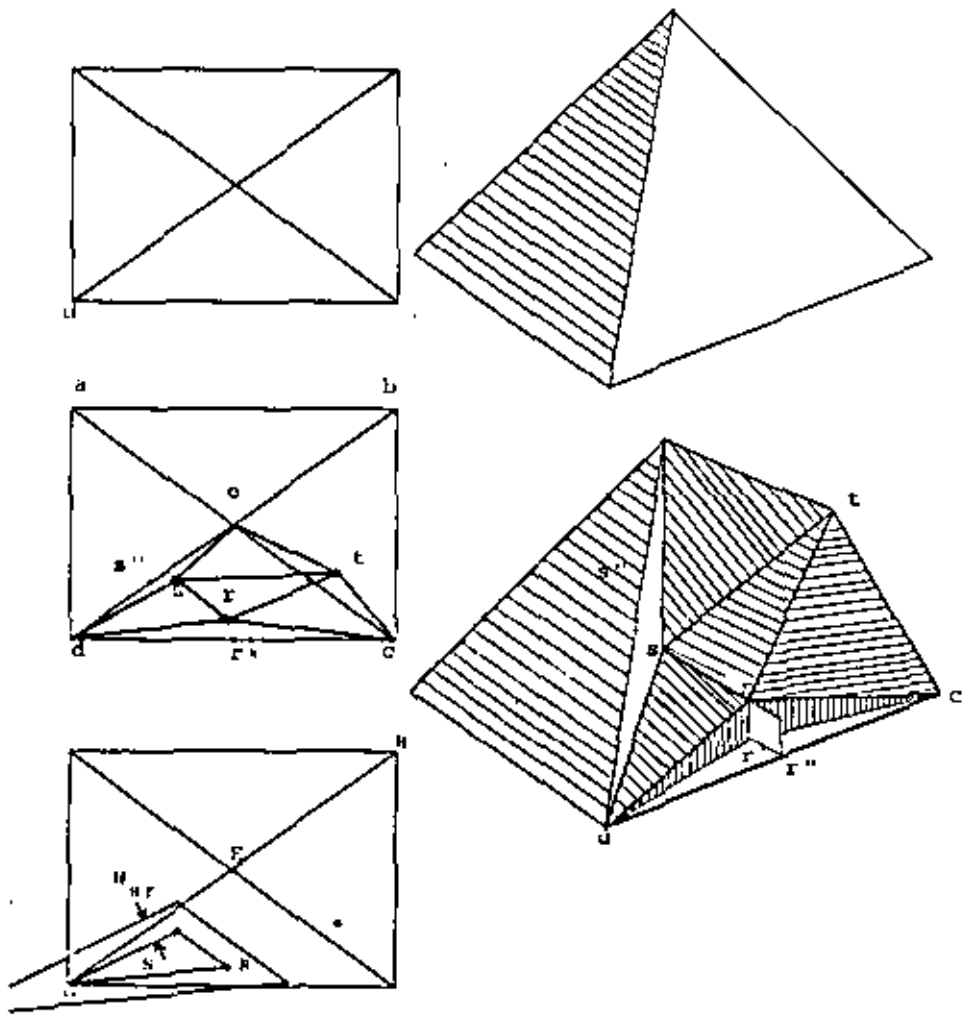


Fig. 7. COPLANAR SIDEWALKS. Compare with Figure 4. If point r = $(r_x, r_y, 0)$ does not fall on line $(c_x, c_y, 0) - (d_x, d_y, 0)$, an horizontal area c d r' will be without coverage by the triangles; a corresponding part of the 3-d surface will fail to be represented. The cure for this is to give "flaps" to the triangles, so that triangle t d i (and its other three brothers) are enlarged by a coplanar sidewalk that covers up to r

```
Procedure ZETA (v1, v2, v3, x, y, m) is defined as
v4 := model [1,m];
  v5 := model [2,m];
    v6 := model [3,m];
      mm := model [4,m];
        n := model [5,m];
         o := model [6,m];
          p := model [7,m];
ZETA := if inside (v1,v4,v6,x,y)
        then    if mm=0 then height (v1,v4,v6,x,y)
                else ZETA (v1,v4,v6,x,y,mm)   % see Table II 'Naming Conventions'
        else if inside (v2,v5,v4,x,y)
        then    if n=0 then height (v2,v5,v4,x,y)
                else ZETA (v2,v5,v4,x,y,n)
        else if inside (v3,v6,v5,x,y)
        then    if o=0 then height (v3,v6,v5,x,y)
                else ZETA (v3,v6,v5,x,y,o)
        else if p=0
        then    height (v1,v4,v6,x,y)
        else    zeta (v1,v4,v6,x,y,p);
END ZETA.
```

The search for the correct triangle that represents a point generates no backtracking. At each level of the tree of triangles, we simply go down to the next level through the appropriate son (that son containing the point), until we hit a terminal triangle, where we compute the height by a planar interpolation.

CONCLUDING REMARKS

Since a gray level picture can be seen as a surface in three dimensions, z being the gray level value, it is in principle possible to use the models described here to represent them. This could have use for shape comparison of these surfaces, but the authors have not experimented with this. The idea, anyway, is to use models with large ε (large error tolerance, coarse representation) to compare two surfaces; if the models are equal (in some appropriate sense, for instance, the quantized z values agree) then we could afford comparison with a smaller ε (more accurate representation). In this way the shape similarity between any two 3-dimensional surfaces (or any two gray level pictures) can be ascertained. A related paper (Brzozowska and Guzmán, 1978a) develops this idea fully for two-dimensional flat regions (binary pictures). The largest problem

with this approach is to find a normalization procedure (the basic rectangle of (Bribiesca and Guzmán,1978b1) that will produce a unique model for the 3-d case: it is easier to compare canonical models.

The method described in this paper is currently being implemented and tested for representation of topographic surfaces formerly described by their contour lines.

Merging of models into a larger model. If four adjacent surfaces a, b, c, d are represented by models $\underline{a}$, $\underline{b}$, $\underline{c}$, $\underline{d}$, the model of the joint surface $[a,b,c,d]$ is formed by creating a new frame 1 (cf. Fig. 5) which has as non-terminal pointers M_a, M_b, M_c and M_d, pointers to the frames 1 of $\underline{a}$, $\underline{b}$, $\underline{c}$ and $\underline{d}$.

Significant points vs. correlation points. The significant points (also called surface-specific points (Peucker, et. al.,1976) are those points of the terrain where slope changes in an important way. The points that a correlation routine finds in an easy manner, based for instance in the two pictures of a stereo pair, are called "correlation points:" they are points that are easy to correlate in the pictures, because the gray levels in their neighborhood are quite different from others, hence they can be identified rapidly and unmistakably. But they will not necessarily fall on top of "significant" points.

The components of the model. The model so far described and its construction can be seen as formed by:

- a tessellation of polygons (Gómez,1978) (triangles in this case);
- an accuracy criteria, which tells whether a polygon of the model needs further refinement (in our case, comparison of modelled vs. real heights, cf. section "When to stop refining");
- a procedure to refine the model (in our model, select a significant point near the middle point of a side);
- a manner to store the model (as exemplified in Fig. 6);
- a way to access the model (as seen in section 'Data Retrieval for Surface Reconstruction');
- a method to reconstruct the surface from the model (this is given by the procedure *height* (a,b,c,x,y) evaluated at the appropriate triangle a b c which contains the point $(x,y,0)$; the appropriate definition of containment is encoded in procedure *inside* (a,b,c,x,y), which takes into account, for instance, the "flaps" of Fig. 7).

Suggestions for further work

1. Refer to Fig. 7. Do not use k_1 = 10% for the width of the sidewalks. Compute instead the maximum distance that $(r_x,r_y,0)$ can be from r'' for the enlarged

triangle i d s to meet still the error tolerance ε. This has to do with average slopes of the triangles.

2. Refer to section "Simplified storage for cover of similar triangles". Fully develop the model that uses the representation of each triangle as a string of pairs of binary digits, v.gr., triangle g k j = 00 00 01 (the son N of the son N of the son N of the rectangle).

3. Do not retrieve the triangles from the root of the tree (cf. section "Data retrieval for surface reconstruction") but store them so as to access them by a double binary search on the coordinates of the vertices (Gómez).

4. Consider the methods of this paper and of (Bribiesca and Avilés, 1974; Bribiesca and Guzmán, 1978a) as similar procedures that address data representation at arbitrary accuracy levels, and use them for shape comparison.

ACKNOWLEDGEMENTS

Andrew Clement and T. Peucker gave the triangular idea; Renato Barrera contributed to the concept of a hierarchy of triangles and other good advices. Abel Carreño and Angel García Amaro, of CETENAL, gave good photogrammetric advice. T. Radhakrishnan kindly revised the manuscript.

Work herein reported was partially done under the Joint Research Agreement (IX-1976) between CETENAL and UNAM.

REFERENCES

Bribiesca, E. and Avilés, R. 1974. Codificación en cadenas y técnicas de reducción de información para mapas y dibujos lineales. IBM Latin American Scientific Center (Mexico City), Informe CCAL-74.

Bribiesca, E. and Guzmán, A. 1978a. Shape description and shape similarity measurement for two-dimensional regions. Submitted to Fourth International Joint Conference on Pattern Recognition. Kyoto, Japan.

Bribiesca, E. and Guzmán, A. 1978b. Shape numbers: a notation to describe pure form and to measure resemblance and difference in shape. Computer Science Dept., IIMAS, National University of Mexico. Report PR-78-18.

Gómez, D. Modelos digitales del terreno de precisión variable. B.S. Thesis, Facultad de Ciencias, Universidad Nacional de México (in preparation).

Gómez, D. 1978. Tesselation of triangles of variable precision as an economical representation for DTM's. Proceedings of the Digital Terrain Models Symposium St. Louis, Mo. Available from American Society of Photogrammetry.

Guzmán, A. 1971. Analysis of curved line drawings using context and global information. In Machine Intelligence VI, (D. Michie and B. Meltzer, eds) University of Edinburgh Press. Chapter 20.

Horn, B.K.P. 1970. Shape from shading: a method for obtaining the shape of a smooth opaque object from one view. Ph.D. Thesis, E.E. Dept., M.I.T. Project MAC Technical Report MAC-TR-79.

Jensen, H.H. 1976. Collaboration in Physics within the Nordic countries. Europhysics news 7, 5, pp 1-4.

Klinger, A. and Nikitas, A. Picture decomposition, tree data structures, and identifying directional symmetries as node combinations. Computer Graphics and Image Processing (to appear).

Peucker, T., Fowler, R.J., Little, J.J. and Mark, D.M. 1976. Triangulated irregular networks for representing three-dimensional surfaces. Simon Fraser University, Burnaby, Canada. Technical Report # 10.

Sigmor, C. and Nadler, M. 1978. Une application de la corrélation numérique d'images: la stéréophotogrammetrie automatique. Congres AFCET/IRIA, Reconnaissance des Formes et Traitement des Images. Paris.

centro de educación continua
división de estudios de posgrado
facultad de ingeniería unam

CASOS PRACTICOS EN EL DISEÑO DE SISTEMAS DE INFORMACION

BASES DE DATOS DISTRIBUIDAS

DR. ALEJANDRO BUCHMANN

27 AGOSTO, 1980

BASES DE DATOS DISTRIBUIDAS

* DEFINICION
* ARQUITECTURAS FACTIBLES
* PROBLEMAS A RESOLVER EN SISTEMAS GENERALIZADOS
* CRITERIOS DE DISEÑO
* EJEMPLOS DE SISTEMAS IMPLEMENTADOS Y SUS LIMITACIONES

DEFINICION

BASE DE DATOS DISTRIBUIDA = SISTEMA DE BASE DE DATOS RESIDENTE EN VARIOS NODOS (COMPUTADORAS) DE UNA RED CONECTADA LOGICA Y FISICAMENTE POR MEDIO DE CANALES DE BAJO ANCHO DE BANDA Y CON CONTROL DISTRIBUIDO.

EJEMPLOS DE REDES

- ARPANET
- CYCLADES
- LIMAS

CONSECUENCIAS DE PROPIEDADES DE LA RED

- DEMORAS DE UN ORDEN DE MAGNITUD

AREAS DE APLICACION

- LINEAS AEREAS
- BANCOS
- DISTRIBUIDORAS
- DISEÑO DE INGENIERIA

* SISTEMA INTEGRADO (SDDI)
* SISTEMA FEDERADO HOMOGENEO
* SISTEMA FEDERADO HETEROGENEO

SISTEMAS INTEGRADOS

DBMS COMUN INCLUYE FUNCIONES PARA

* MODIFICACIONES CONCURRENTES
* LOCALIZACION DE DATOS
* RECUPERACION
* INTEGRIDAD
* SEGURIDAD
* MAPEO A VARIOS MODELOS DE DATOS

NO EXISTE ACTUALMENTE UN SISTEMA GENERALIZADO EN OPERACION

SISTEMAS FEDERADOS HOMOGENEOS

FEDERACION DE BASES DE DATOS DISEÑADAS PARA OPERAR INDEPENDIENTEMENTE

COOPERACION POR MEDIO DE INTEGRADOR-SUPERVISOR

MISMO DBMS Y MISMA MAQUINA EN CADA NODO

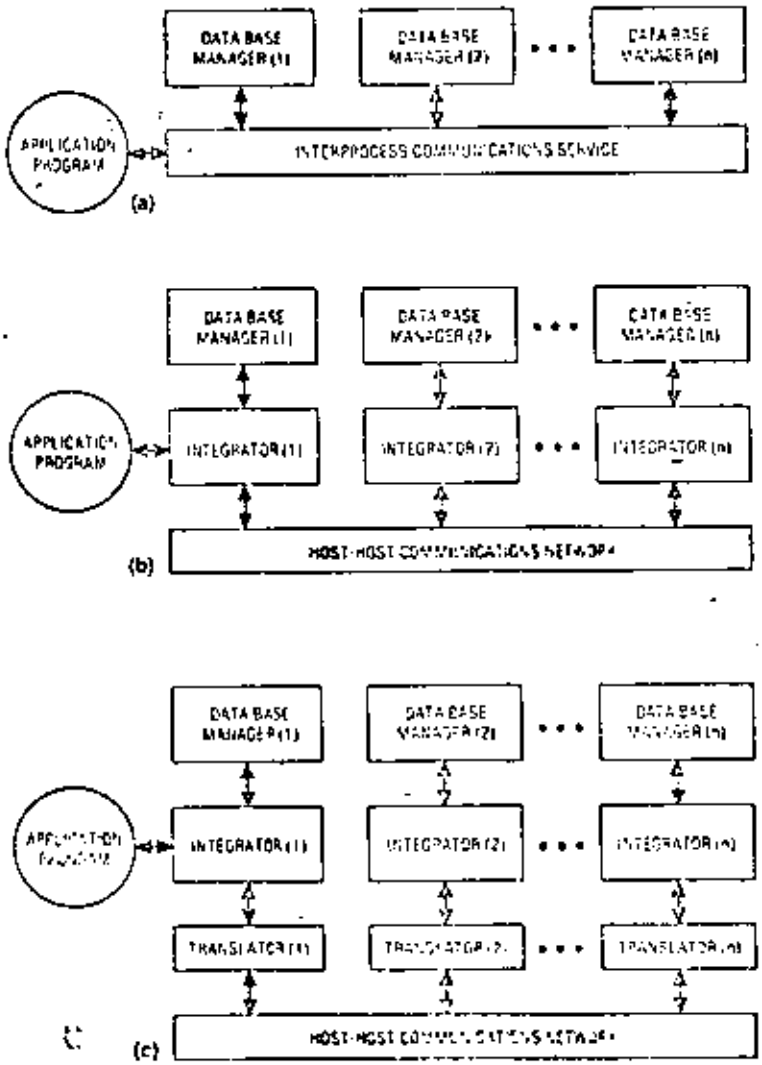
SISTEMA FEDERADO HETEROGENEO

COOPERACION COMO EN EL CASO HOMOGENEO

EQUIPO Y DBMS HETEROGENEO EN CADA NODO

PROBLEMAS DE TRADUCCION DE UN DBMS A OTRO

NO ES IMPLEMENTABLE EN FORMA GENERAL


DATA BASE MANAGER (1)
DATA BASE MANAGER (2)
DATA BASE MANAGER (n)
APPLICATION PROGRAM
INTERPROCESS COMMUNICATIONS SERVICE
(a)
DATA BASE MANAGER (1)
DATA BASE MANAGER (2)
DATA BASE MANAGER (n)
APPLICATION PROGRAM
INTEGRATOR (1)
INTEGRATOR (2)
INTEGRATOR (n)
HOST-HOST COMMUNICATIONS NETWORK
(b)
DATA BASE MANAGER (1)
DATA BASE MANAGER (2)
DATA BASE MANAGER (n)
INTEGRATOR (1)
INTEGRATOR (2)
INTEGRATOR (n)
TRANSLATOR (1)
TRANSLATOR (2)
TRANSLATOR (n)
HOST-HOST COMMUNICATIONS NETWORK
(c)

PROBLEMAS DE SISTEMAS GENERALIZADOS QUE SE ANALIZARAN

* PARTICION DE LA BASE DE DATOS Y SELECCION DE NODOS DE RESIDENCIA

* LOCALIZACION DE LOS DATOS

* CONTROL DE ACCESOS CONCURRENTES

* SEGURIDAD

* INTEGRIDAD

DISTRIBUCION DE LOS DATOS

I DISTRIBUCION LOGICA - DISTRIBUCION FISICA

DISTRIBUCION LOGICA

* DISTRIBUCION ES PARTE DEL ESQUEMA VISIBLE AL USUARIO
* USUARIO DIRIGE PREGUNTA POR DATO X AL NODO Y

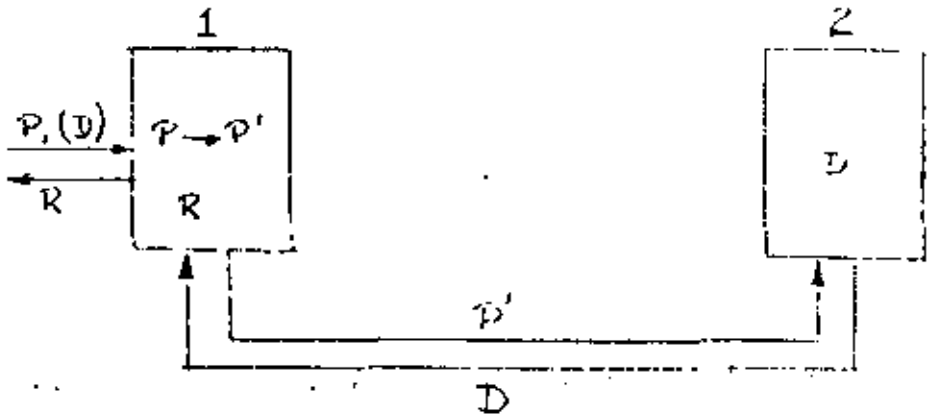
EJEMPLOS: PARTICION GEOGRAFICA, POR CIUDAD DE RESIDENCIA CLIENTE

DISTRIBUCION FISICA

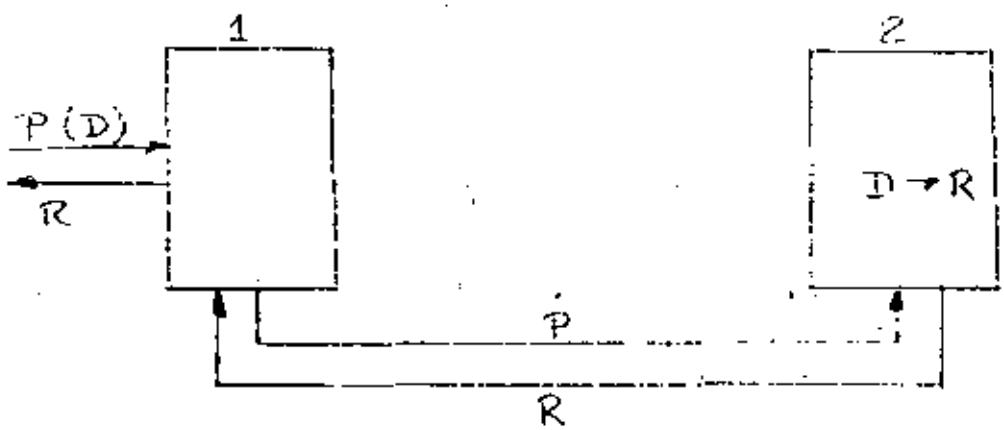
* INVISIBLE AL USUARIO
* PUEDE PERO NO TIENE QUE SEGUIR PARTICION LOGICA
* SISTEMA DISTRIBUYE DATOS

EJEMPLO: SISTEMA DISTRIBUYE DATOS A GUADALAJARA, MONTERREY Y MEXICO. SI FALLA GUADALAJARA DATOS PUEDEN ALMACENARSE TEMPORALMENTE EN MONTERREY SIN QUE EL USUARIO LO NOTE

II DATOS A LOS PROCESOS VS. PROCESOS A LOS DATOS



CONVENIENTE SI $P' \ll P$
$R \gg D$
P' FACILMENTE EXTRAIBLE DE P



CONVENIENTE SI $R \ll D$
P ES PEQUEÑO
$P+R \ll D$

FACTORES DETERMINANTES PARA DISTRIBUCION

* USO
* COSTO COMUNICACION / TRANSMISION DE DATOS
* TIEMPO DE RESPUESTA MAXIMO PERMISIBLE
* PORCENTAJE DE ACTUALIZACIONES

USO DE METODOS HEURISTICOS (PROBLEMA ANALITICO ES EXPONENCIAL)

TENDENCIA DE COSTSOS Δ COSTO PROCESADO $\approx$ 2 x Δ COSTO COM.

CRITERIO

$U / A < 0.1$ MULTICOPIAS

$U / A > 0.1$ COPIA UNICA SEGMENTADA

LOCALIZACION DE LOS DATOS

* INDICE COMO PARTE DEL IDENTIFICADOR
  - PROBLEMATICO REDISTRIBUIR DATOS

* DIRECTORIO COMPLETO CENTRALIZADO
  + POCO OVERHEAD DE DIRECTORIO
  - VULNERABILIDAD
  - INCREMENTO DE TRAFICO

* DIRECTORIO COMPLETO EN CADA NODO
  + PROCESAMIENTO EFICIENTE
  + REDUCCION COSTO DE TRANSMISION
  - OVERHEAD EN CADA NODO
  - COSTOSO MODIFICAR DIRECTORIO

* BROADCAST
  - OVERHEAD POR INVOLUCRAR EN CADA SOLICITUD
    PROCESADORES SUPERFLUOS

* OPCION PRACTICA = VARIOS DIRECTORIOS COMPLETOS

LOCALIZADOS EN NODOS ESTRATEGICOS

* CRITERIOS

DIRECTORIO COMPLETO DISTRIBUIDO SI $U < 0.1\ A$

DIRECTORIO COMPLETO CENTRALIZADO SI $U > 0.1\ A$

CONTROL DE CONCURRENCIA

DEPENDE DE DISTRIBUCION DE DATOS (U/A)

ACCESOS DE LECTURA NO CAUSAN PROBLEMAS

ACCESOS DE MODIFICACION (UPDATE) PUEDEN PROVOCAR INCONSISTENC

Y PUNTO MUERTO (DEADLOCK)

SOLUCIONES AL PROBLEMA DE PUNTO MUERTO

* EVITAR PUNTO MUERTO
* DETECTAR PUNTO MUERTO Y REINICIAR

DEFINICION PUNTO MUERTO

ES LA SITUACION QUE OCURRE SI DOS PROCESADORES A Y B HAN INICIADO UNA TRANSACCION Y NO PUEDEN TERMINARLA, YA QUE A REQUIERE RECURSOS BLOQUEADOS POR B Y VICEVERSA.

DETECCION Y CORRECCION DE PUNTO MUERTO

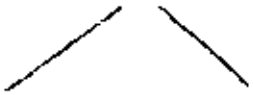
* ESPERAR A QUE OCURRA PUNTO MUERTO
* DETECTAR PUNTO MUERTO MONITOREANDO PROGRESO DE CADA TRANSACCION
* ABORTAR LA TRANSACCION MAS RECIENTE
* DEJAR TERMINAR LA OTRA TRANSACCION
* REINICIAR SEGUNDA TRANSACCION

PREVENCION DE PUNTO MUERTO

* ESCOGER GRANULARIDAD FINA --- OVERHEAD ADMINISTRATIVO

  GRUESA - DEGRADACION DEL SISTEMA

* UNICA ESTRATEGIA QUE GARANTIZA CONSISTENCIA ES LA DE CHANDY



OK

XX

L-L-L-U-U-U

L-L-U-L-U-U

* PARA SISTEMAS ESPECIALIZADOS CON POCAS TRANSACCIONES DIFERENTES SE PUEDE PREDEFINIR UNA SECUENCIA DE BLOQUEO

LIMITACIONES PRACTICAS A LA GRANULARIDAD

CONTROL CENTRAL — VULNERABILIDAD

RIGIDEZ

CONTROL LOCAL — CADA NODO EMITE REPORTES

CADA NODO RECONSTRUYE DE ESOS REPORTES EL ESTADO DE LA BASE DE DATOS

OVERHEAD INTOLERABLE EN AMBOS CASOS SI LA GRANULARIDAD ES MUY FINA

PROBLEMAS DE INTEGRIDAD

CAUSAS

* CONTROL DE CONCURRENCIA INADECUADO
* SW CON ERRORES
* FALTAS EN LA SEGURIDAD
* CAIDAS DEL SISTEMA

CONTROL DE CONCURRENCIA INADECUADO

SECUENCIA CORRECTA DE TRANSACCIONES EN UN BANCO

1) SALDO INICIAL = 50,000

2) DEPOSITO EN EFECTIVO SUC. A = 100,000

3) CHEQUE PORTADOR PARA RETIRAR 100,000 EN SUC. B

SI 3 OCURRE ANTES QUE 2 CHEQUE REBOTA, CAUSA MULTA, ETC.

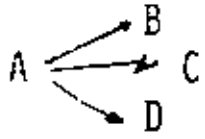
FALLAS DE SEGURIDAD

DATO X TIENE DIFERENTE PROTECCION EN NODO A Y B

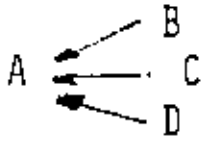
PROBLEMA COMUN EN SISTEMAS HETEROGENEOS

CAIDAS DEL SISTEMA

ACTUALIZACION CON MULTIPLES COPIAS

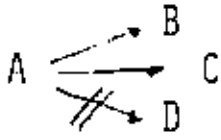


EFECTUA ACTUALIZACION Y AVISA



ACTUALIZACION REALIZADA PERO NO EN EFECTO

TODOS TRABAJAN CON VALOR VIEJO



PONER EN EFECTO

B Y C TRABAJAN CON VALOR NUEVO, D CON VALOR VIEJO

NO EXISTE SISTEMA GENERALIZADO OPERACIONAL

ALTERNATIVA:

PODEMOS IMPLEMENTAR UN SISTEMA DISTRIBUIDO SI:

* BASADO EN PROPIEDADES DE LA APLICACION PODEMOS EVITAR

  PROBLEMAS

* SI CONSECUENCIAS DE UNA INCONSISTENCIA SON DE POCA

  IMPORTANCIA

* SI TRANSACCIONES SON LIMITADAS O PRINCIPALMENTE DE LECTURA

SITA (SOCIETE INTERNATIONALE DE TELECOMMUNICATIONS AERONAUTIQUES)

ARINC (AERONAUTICAL RADIO INC.)

APLICACION: RESERVACIONES AEREAS-INTERLINEA

CONFIGURACION: ESTRELLA

CADA LINEA TIENE PROPIA BASE DE DATOS

PRIMER VUELO DEFINE LA BASE DE DATOS DE ACCESO

VUELOS SUBSECUENTES SE ENVIAN A SITA o ARINC

NODO CENTRAL TRADUCE TRANSACCION Y LA ENVIA AL NODO DE LA LINEA CORRESPONDIENTE

LIMITACIONES: TRANSACCIONES PERMITIDAS: RESERVACION
CANCELACION
- STATUS
LISTA DE ESPERA

"OVERBOOKING"

SI CAE NODO CENTRAL MUERE TODO EL SISTEMA
(SOLUBLE POR MEDIO DE REDUNDANCIA TOTAL EN NODO CENTRAL)

MENSAJES STANDARD FACILMENTE CONVERTIBLES

PREDEFINICION DEL ORDEN DE BLOQUEO

CASO TIPICO DE SISTEMA SEGMENTADO

BASE DE DATOS DEL BANK OF AMERICA

APLICACION: CONTROL DE 11 MM DE CUENTAS EN MAS DE 1000 SUCURSALES

ORIGINALMENTE SISTEMA BATCH

PERDIDAS ANUALES DE MAS DE 3 MM$

CONFIGURACION: DOS NODOS TOTALMENTE REDUNDANTES EN LA Y SF

SISTEMA PERFECTAMENTE HOMOGENEO

PARTICION GEOGRAFICA, E.D. TRANSACCIONES SE PROCESAN EN EL NODO QUE TIENE LOS DATOS Y ESTOS SE DISTRIBUYEN POR PROXIMIDAD DE LA SUCURSAL CORRESPONDIENTE AL NODO.

METAS QUE SE LOGRARON: 33% DE DISMINUCION DE PERDIDAS

40% REDUCCION DE GASTOS TELEFONICOS

16% REDUCCION PERSONAL

VENTAJAS ESTIMADAS SOBRE SISTEMA CENTRALIZADO

40% AHORRO EN DESARROLLO

3 MM $ AHORRO COSTO OPERACION

LOWES COMPANIES INC.

APLICACION: CONTROL DE INVENTARIOS, CUENTAS Y VENTAS PARA UNA FERRETERIA CON 140 SUCURSALES

CONFIGURACION: ESTRELLA

NODO CENTRAL EN EL CORPORATIVO

NODO EN CADA SUCURSAL CONSISTE DE UNA MINI

Y UN MAXIMO DE 16 TERMINALES

BASE DE DATOS TOTALMENTE SEGMENTADA

LIMITACIONES: CADA BASE DE DATOS FUNCIONA EN FORMA LOCAL

NO HAY COMUNICACION ENTRE NODO PERIFERICOS

COMUNICACION NOCTURNA EN FORMA DE BATCH CON EL NODO CENTRAL

COMUNICACION NOCTURNA EN FORMA DE BATCH DEL NODO CENTRAL A LOS PERIFERICOS

COMENTARIO: NO MERECE EL NOMBRE DE BASE DE DATOS DISTRIBUIDA.

ES UNA COLECCION DE BASES DE DATOS CENTRALES QUE GENERA REPORTES ENTENDIBLES A OTRA BASE DE DATOS Y RECIBE INSTRUCCIONES EN LA MISMA FORMA

RSEXEC DE ARPANET

APLICACION: ABIERTA

CONFIGURACION: RED GENERALIZADA

LIMITACIONES: NO ES UN SISTEMA DE BASE DE DATOS SINO UN PAQUETE DE UTILERIA PARA MANEJO DE ARCHIVOS PERO DE FORMATO Y BAJO PROTOCOLOS ACEPTABLES A TODOS LOS NODOS EN LA RED.

PUBLICACIONES

1. "Una generalización del teorema de Aronhold-Kennedy", Revista INGENIERIA, vol. XL, No. 1, enero de 1970.

2. "Algunos teoremas relativos al análisis de mecanismos en tres dimensiones", Revista INGENIERIA, vol. XL, No. 3 julio-septiembre de 1970.

3. "Initial-Boundary Value Problems in Viscoplastic Media", Tesis doctoral, Departamento de Mecánica Aplicada, Universidad Stanford, 1973.

4. "Computer-Aided Analysis and Synthesis of Cam Mechanisms", Trabajo presentado ante el II Congreso Internacional de Sistemas e Informática (CIASI). Noviembre de 1974, México, D.F.

5. "Educación en Ingeniería Mecánica en México a Nivel de Licenciatura", Ponencia presentada ante la VI Asamblea Bienal del Colegio de Ingenieros Mecánicos Electricistas, Noviembre de 1974, Puebla, Pue.

6. "Procesos Dinámicos en Medios Viscoplásticos", Revista INGENIERIA, vol. XLIV, No. 4, octubre-diciembre de 1974.

7. "Análisis y Síntesis Cinemáticos de Sistemas Mecánicos", Editorial Limusa, México, 1978.

8. "Estudios de Especialización y Grado en Mecánica Aplicada", Serie Orientación, No. 3, Publicación del Consejo Nacional de Ciencia y Tecnología, México, 1975.

9. "Un Currículum de Ingeniería Mecánica y una Estrategia para su Implantación en Instituciones Nacionales", coautor con Alejandro F. Romero, Memoria del I Congreso de la Academia Nacional de Ingeniería, Guanajuato, junio de 1975.

10. "Programas de posgrado en Ingeniería Mecánica e Ingeniería Eléctrica Conducentes al Grado de Maestro en Ingeniería, en la Universidad de Guanajuato", Coautor con Arturo Lara López, Memoria del I Congreso de la Academia Nacional de Ingeniería, Guanajuato, junio de 1975.

11. "Dynamic Analysis of the inflation of a Viscoplastic Hollow Sphere", Proceedings of the VIII Southeastern Conference on Theoretical and Applied Mechanics, Blacksburg, Virginia, 29 y 30 de abril de 1976.

12. "Un Modelo Lineal por Tramos para Simular una Clase de Procesos en Medios Viscoplásticos", Revista INGENIERIA, vol. XLVI, No. 2, abril-junio de 1976.

13. "Necesidad de Formación de Recursos Humanos para la Industria de Bienes de Capital", trabajo presentado ante el Seminario sobre el Costo de la Tecnología, organizado por la Sociedad Mexicana de Ingeniería Económica y de Costos, C. de México, junio de 1976.

14. "Optimización Cuadrática de Mecanismos con Pares Inferiores" (con C. Palacios), Memoria del II Congreso de la Academia Nacional de Ingeniería, A.C., Monterrey, N.L., septiembre de 1976.

15. "Optimal Synthesis of Cam Mechanisms via the Methods of Newton-Raphson and Runge-Kutta", Second IFToMM International Symposium on Linkages and Computer-Aided Design Methods, Bucarest (Rumania), 16-21 de junio de 1977.

16. "Matrix Methods in Applied Kinematics", División de Estudios Superiores de la Facultad de Ingeniería, UNAM, 1978.

17. "Alcances y Limitaciones de los Modelos Lineales en la Mecánica", Revista INGENIERIA, vol. XLVIII, No. 4, pp.292-297.

18. "Numerical solution to the Input-output Displacement Equation of the General 7R Spatial Mechanism", (con H. Albala) Memoria del V Congreso Mundial de Mecanismos y Teoría de la Máquinas, Montreal, julio 9-13, 1979.

19. "Optimal Synthesis of Linkages using Householder Relfections", Memoria del V Congreso Mundial de Mecanismos y Teoría de las Máquinas, Montreal, julio 9-13, 1979.

20. "Programa para Analizar Digitalmente el Desplazamiento del Mecanismo Espacial 7R General", (con A. Rojas) Memoria del IV Congreso de la Academia Nacional de Ingeniería, A.C., Mérida, Yucatán, 11-13 de octubre de 1978.

21. "Diseño Optimo de Mecanismos de Leva con Seguidor de Rodillo" (con R. Reyes) Memoria del IV Congreso de la Academia Nacional de Ingeniería, A.C., Mérida, Yucatán, 11-13 de octubre de 1978. Es un resumen de la Tesis de Licenciatura del coautor.

22. "Modelo Dinámico de una Suspensión para Vehículos de Transporte Masivo", presentado en el Mexicon 78, Cd. de México, 2-5 de agosto de 1978.

23. "Suspension-system synthesis for mass transport vehicles with prescribed dynamic behaviour", enviado a: Journal of Mechanical Design, Trans. ASME. Coautor: I. Espinosa

24. "On the Modification of the local geometric properties of plane curves", aceptado para su publicación en la revista Computer-Aided Design, Londres. Aparecerá en 1980.

25. "Manual zur Anwendung der Subroutinen zur Darstellung ebener geschlossenen Kurven und zur Steuerung ihrer lokalen Eigenschaften" (Manual del Usuario para la Aplicación de las subrutinas empleadas en la representación de curvas planas cerradas y en el control de sus propiedades locales), Reporte interno del Laboratorio de Máquinas Herramienta del Instituto Tecnológico Renano-Westfálico de Aquisgrán, RFA, 1979. Coautor: P.Steinke.

26. "Software zur Synthese von ebenen geschlossenen Kurven im Zusammenhang mit der Optimierung von Bauteilen" (Subprogramas para la síntesis de curvas planas cerradas en relación con la optimización de elementos de máquinas), para enviarse a la revista Konstruktion, de la RFA. Coautor: P. Steinke.

En elaboración:

27. "On the control of the global geometric properties of plane cuerves", para enviarse a la revista Computer-Aided Design, Londres.

28. "On the computation of the screw parameters of a rigid-body motion. Part I: Finitely separated positions", para enviarse a la revista Mechanism and Machine Theory, Londres.

29. "On the computation of the screw parameters of a rigid-body motion. Part II: Infinitely separated positions", para enviarse a la revista Mechanism and Machine Theory, Londres.

30. "An optimality criterion to supress stress concentrations in linearly elastic bodies", para enviarse a la revista Acta Meccanica, Viena.

31. "Über die Bestimmung der optimalen Kontur eines Kettengliedes zur Verminderung von Spannungsspitzen", para enviarse a la revista Konstruktion, RFA

centro de educación continua
división de estudios de posgrado
facultad de ingeniería unam

CASOS PRACTICOS EN EL DISEÑO DE SISTEMAS DE INFORMACION

Descripción de un Sistema para Procesamiento de Información Geográfica

Ing. Ernesto Bribiesca Correa

Septiembre , 1980.

NACIONES UNIDAS

CONSEJO ECONOMICO Y SOCIAL

Distr.
Limitada

E / Conf. 71 / L. 39
30 Septiembre 1979

ORIGINAL: ESPAÑOL

SEGUNDA CONFERENCIA CARTOGRAFICA REGIONAL DE LAS NACIONES UNIDAS PARA AMERICA
México, D. F., 3 a 14 de Septiembre de 1979
Tema 8a del programa provisional

DESCRIPCION DE UN SISTEMA PARA PROCESAMIENTO DE INFORMACION GEOGRAFICA*

(Documento presentado por México)

* Preparado por: Ing. Ernesto Briblesca Correa.
DETENAL

DESCRIPCION DE UN SISTEMA PARA PROCESAMIENTO DE INFORMACION GEOGRAFICA

Por: Ernesto Bribiesca C.

RESUMEN

La Dirección General de Estudios del Territorio Nacional tiene como objetivo principal el inventario de los recursos naturales y potenciales del país, proporcionando esta información a las dependencias oficiales y entidades particulares que así lo soliciten para llevar a cabo el armónico desarrollo del país, por medio de una adecuada planeación.

Así la información producida es rica y abundante, por lo cual se hace recomendable el uso de los dispositivos electrónicos de cómputo para su procesamiento; en este trabajo se describe un sistema llamado SICA (Sistema Integral de Cartografía Automatizada) para el manejo masivo de información geográfica.

El SICA interrelaciona información geográfica y socio-económica para: la elaboración de mapas derivados, la toma de decisiones y la planeación.

DESCRIPCION DE UN SISTEMA PARA PROCESAMIENTO DE INFORMACION GEOGRAFICA

Por: Ernesto Bribiesca C.

CONTENIDO

I INTRODUCCION

II DESCRIPCION

III DISEÑO DEL SISTEMA

CONCLUSIONES Y RECOMENDACIONES

REFERENCIAS

I. INTRODUCCION

El Sistema Integral de Cartografía Automatizada (SICA) es una herramienta para el proceso de datos geográficos, en la generación de mapas derivados a partir de información de mapas básicos.

En el diseño de las bases de datos geográficos, se usan generalmente dos tipos de estructuras: estructuras reticulares a base de celdas, y estructuras poligonales a base de ventanas, arcos y nodos como elementos de una red. El SICA es un sistema de estructura hf brida, el cual usa las ventajas de las estructuras reticulares y poligonales aprovechando las ventajas de cada una de ellas: la estructura reticular tiene la ventaja de un rápido acceso a los datos y la estructura poligonal tiene la ventaja de una precisión aceptable.

El SICA esta diseñado para producir mapas derivados a la misma precisión con que son digitalizados los mapas básicos.

II. DESCRIPCION

Dada una región cualquiera que se desee accesar al sistema se le considerará como el universo físico; este universo físico se puede describir cualitativamente y cuantitativamente en algunas de sus características por ejemplo: la química del suelo, los tipos de rocas, la geomorfología, las clases de vegetación, la magnetometría, la energía reflejada, etc; cada una de estas características se asociarán como temas; los temas ahora considerados en el sistema son aquellos que nos describen características asociadas a procesos taxonómicos, o sea información que ya ha sido analizada e interpretada por un especialista.

Así el universo físico queda descrito en diferentes temas dependiendo de los estudios que se quieran realizar en el mismo. En el sistema cada tema tiene propiedades excluyentes, es decir, dentro del mismo tema existe una propiedad u otra pero no ambas, por ejemplo: En el tema de Geología las rocas metamorficas, riolitas y sedimentarias son representadas en un documento cartográfico en forma excluyente; además cada una de estas propiedades esta cualificada y ubicada en espacio y tiempo, en el sistema esto se conoce como evento natural.

Al SICA se le pueden accesar tanto diferentes temas como sean necesarios, cada uno con propiedades excluyentes como eventos naturales definidos por su propiedad, coordenadas y fechas de actualización, es lógico suponer que cualquier clase de información de índole cartográfico o socioeconómico puede ser accesada, por ejemplo: área de agricultura de riego, número de estudiantes que padecieron más de tres caries en la escuela "Lic. Benito Juárez" en 1970, líneas de Energía Eléctrica, etc.

La información almacenada en el SICA es información DETENAL en sus diferentes temas. Así, el universo físico será descrito con los temas usados por DETENAL, y las aplicaciones y usos serán a través de estas variables. El sistema tiene el compromiso con el usuario de proporcionarle ésta información en forma rápida, confiable y combinada para la solución adecuada de sus necesidades.

III DISEÑO DEL SISTEMA

Con las definiciones del capítulo anterior se simplifica el estudio del diseño del sistema; como se vió el universo físico puede ser descrito en varios temas, cada uno con sus eventos naturales excluyentes definidos por su propiedad o característica en el espacio y en el tiempo (ver figura Descripción del Universo Físico en temas de interés).

CAPTURA DE DATOS.

Una vez seleccionados los temas de interés para la descripción del Universo Físico; estos deben ser representados digitalmente para ser introducidos a la base de datos del sistema. A un tema se le asigna su clave y se digitalizan los contornos de las propiedades, no importando el orden de las mismas. Esto genera sus coordenadas (x, y) o (Ψ , λ) a la precisión deseada, a continuación se digitaliza un punto dentro del área asignandole su propiedad (ver figura digitalización de propiedades), de esta forma cada tema es digitalizado y almacenado en cinta magnética.

GEOLOGIA

Rocas Igneas

D - Diorita

R - Riolita

Rocas Sedimentarias

cz - Caliza

Rocas Metamórficas

P - Pizarra

EDAFOLOGIA

Unidades de Suelo

L - Luvisol

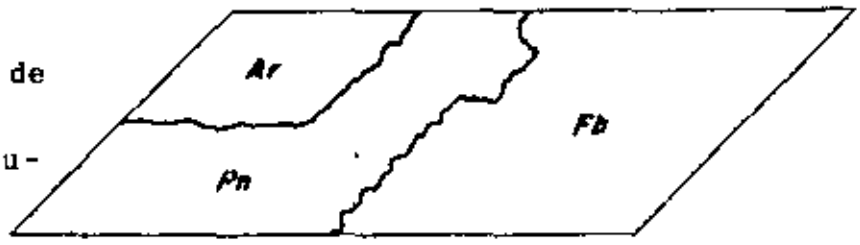
N - Nitosol

W - Planosol

USO DEL SUELO

Ar - Agricultura de Riego.

Pn - Pastizal natural.



VIAS DE COMUNICACION

El - Energía Eléctrica

cp - Carretera pavimentada

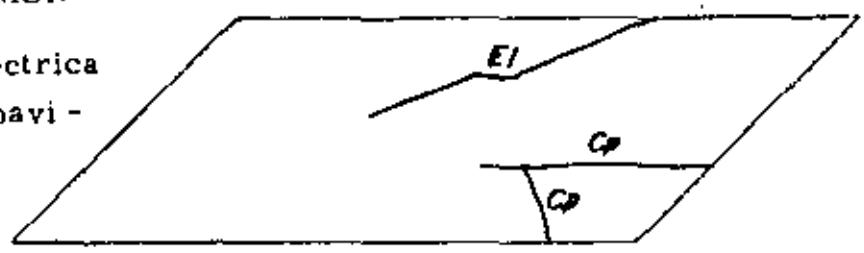


Figura. Descripción del Universo Físico en temas de Interés.

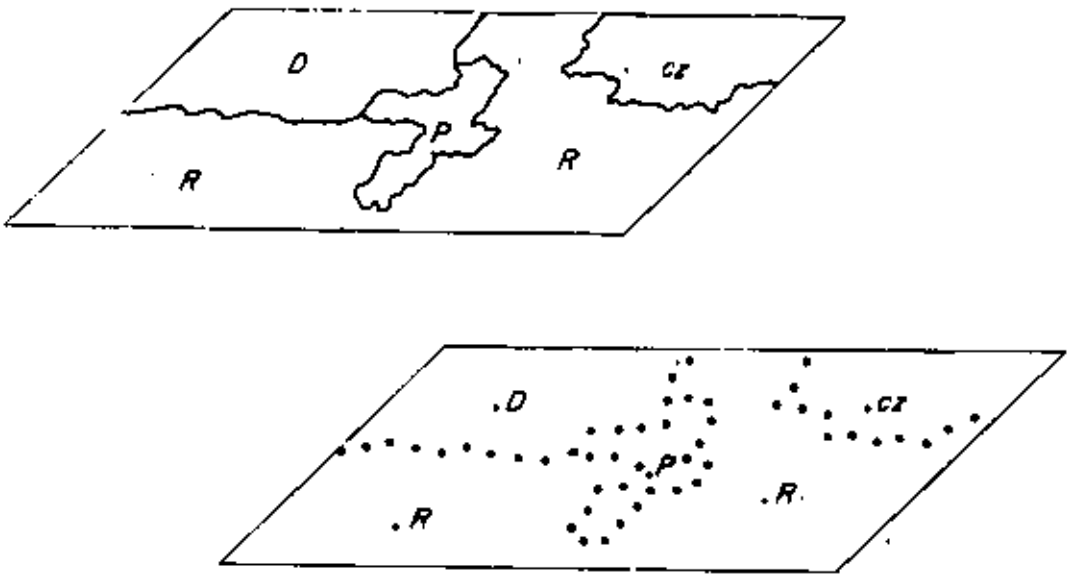


Figura Digitalización de Propiedades.

PROCESO DE DATOS

La información digitalizada es leída y dividida en dos archivos; llamados el archivo de matrices y el archivo de líneas. El archivo de matrices o mapas numéricos es generado tomando un tema y normalizandolo al tamaño de una matríz(el tamaño de la matriz lo define la menor área digitalizada, que será el elemento de resolución de la matriz) etiquetando cada contorno. Ver figura normalización de un tema.

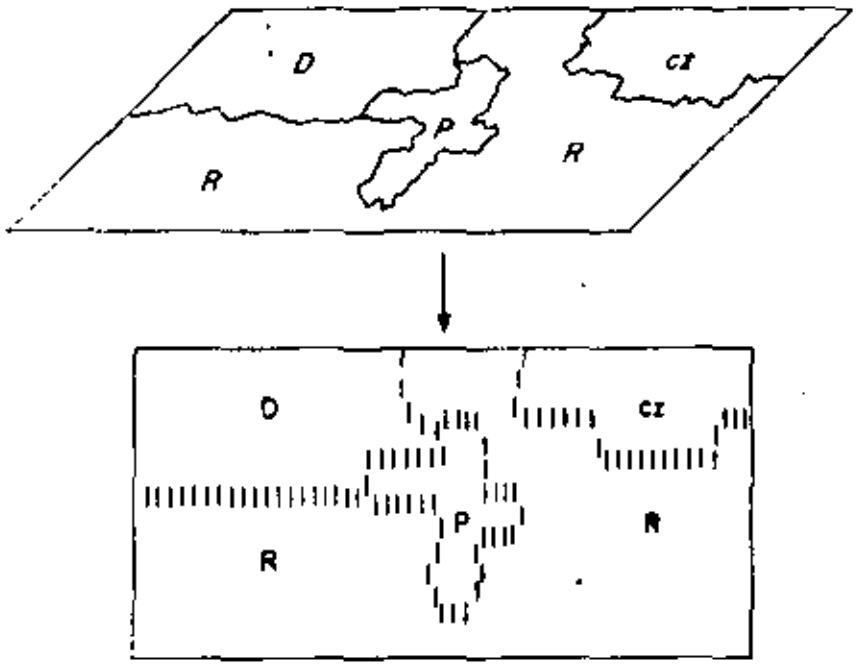


Figura Normalización de un Tema.

Una vez normalizadas las propiedades de un tema en la matríz se procede a "pintarla", que consiste en extender su propiedad hasta el contorno que la define, el algoritmo usado es una rutina llamada "Torre de Ajedréz" que "pinta" usando un movimiento ortogonal; los contornos marcados con 1's son eliminados promediando las propiedades de los vecinos. (Ver figura Generación del mapa numérico y figura Mapas Numéricos de los temas de Interés).

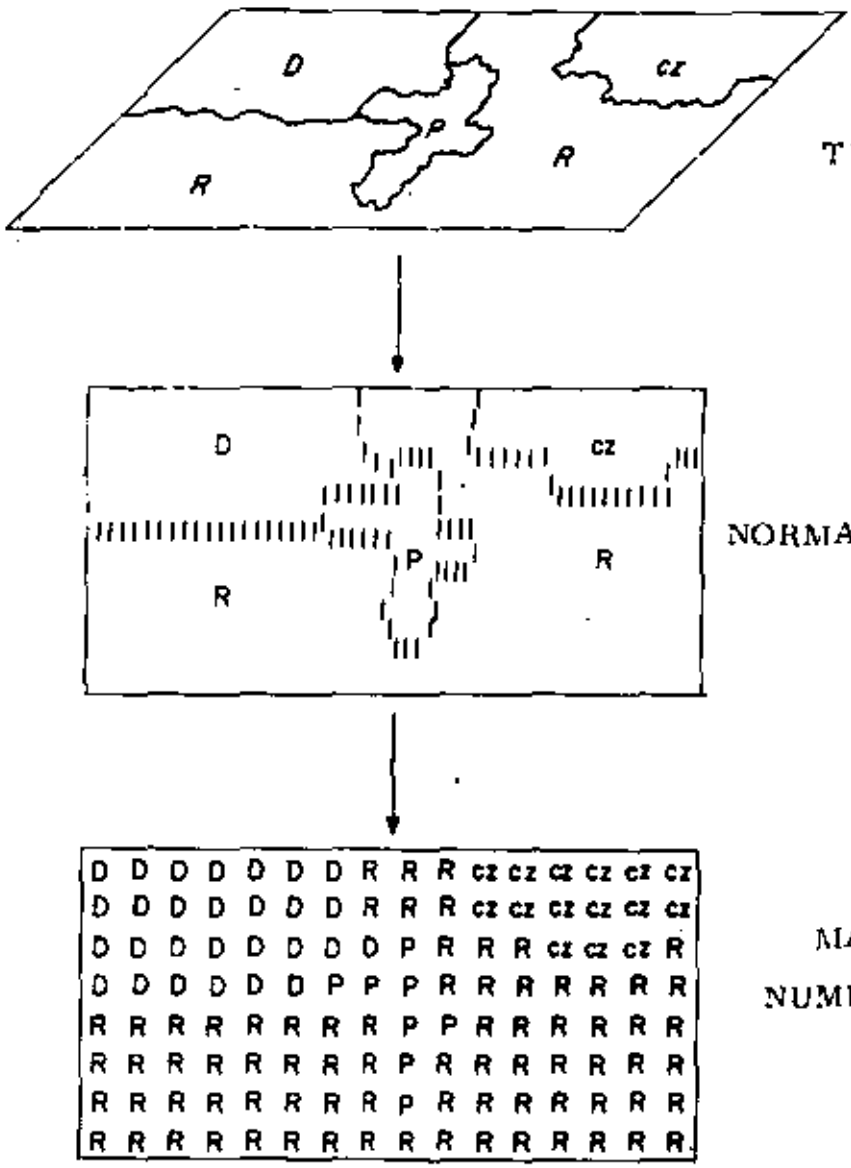
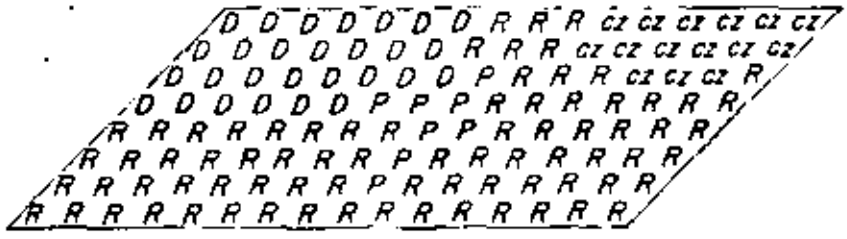


Figura Generación del Mapa Numérico

Los mapas numéricos son los que constituyen el archivo de matrices; cierta clase de información, debe tener otros preprocesos, la información topográfica representada por curvas de nivel, es trans - formada a un modelo digital del terreno usando interpolación, esto a su vez ya constituye un mapa numérico. Las redes de caminos y vías de

GEOLOGIA

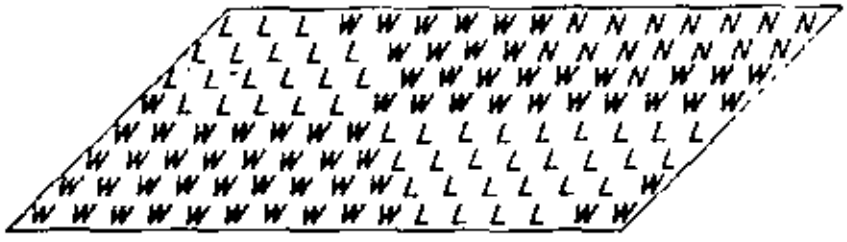
Rocas Igneas.
D - Diorita
R - Riolita
Rocas Sedimentarias.
cz - Caliza
Rocas Metamórficas.
P - Pizarra.



EDAFOLOGIA

Unidades de Suelo

L - Luvisol.
N - Nitosol.
W - Planosol.



USO DEL SUELO

Ar - Agricultura de Riego
Pn - Pastizal natural.
Fb - Bosque natural.

VIAS DE COMUNICACION

El - Energía Eléctrica.
Cp - Carretera Pavimentada.

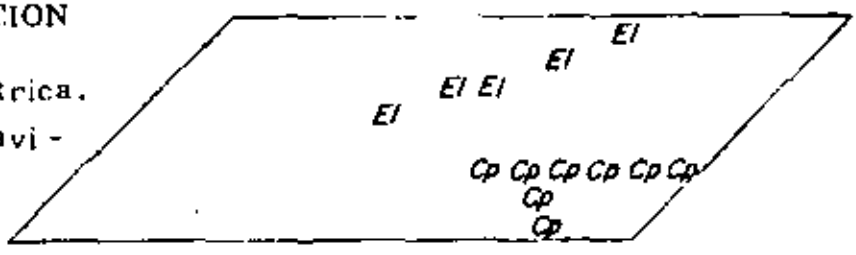


Figura . Mapas Numéricos de los temas de Interés.

comunicación son accesadas directamente. Así cualquier tipo de información ubicada en espacio y tiempo es digitalizada, normalizada y transformada a un mapa numérico. El archivo de líneas se obtiene vaciando la información digitalizada al mismo, este archivo se usa al final del proceso, para recuperar la precisión de la información.

GENERACION DEL ARBOL RECONFIGURABLE

Ya con los temas representados por mapas numéricos se puede describir el universo físico a determinada precisión, ahora el sistema generará su propio árbol de descriptores que permitirá al usuario tener una rápida respuesta a sus consultas. Para reconstruir el árbol reconfigurable, se genera la raíz del árbol haciendo una lista de todas las propiedades consideradas en los diferentes temas del universo físico (ver figura Generación del árbol reconfigurable), esto es a partir del archivo de mapas numéricos, el universo físico es dividido en cuatro partes iguales, llamadas los hijos, en cada hijo se coloca también una lista de todas las propiedades de todos los temas considerados y contenidos en el mismo, así sucesivamente se siguen dividiendo los hijos y marcando sus propiedades, hasta que una propiedad alcance el 100%, o solamente existan dos propiedades excluyentes del mismo tema o de otros; cuando sucede esto se consulta el archivo de líneas, las cuales

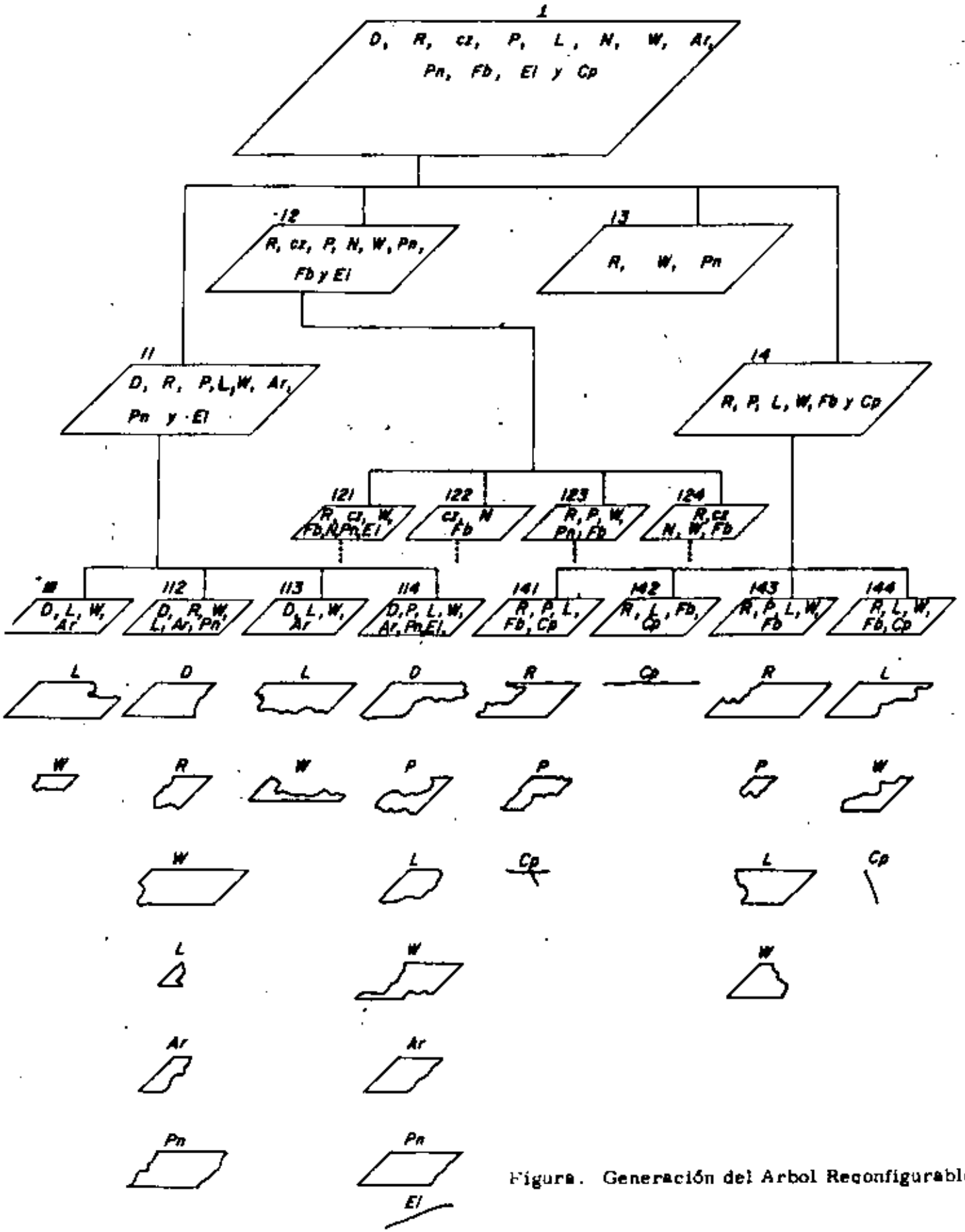


Figura. Generación del Arbol Reconfigurable

son "rociadas" sobre esta estructura reticular no uniforme (de celdas grandes y pequeñas), donde son almacenadas en cada hijo correspondiente: aquí es donde se almacena la precisión original con que se digitalizó la información.

La raíz del árbol es etiquetada con 1, sus hijos o sean sus cuatro ramas con 11, 12, 13 y 14 respectivamente, y así sucesivamente por todo el árbol.

La generación del árbol reconfigurable es la parte más interesante del sistema, ya que aquí se conjugan diferentes estructuras, la estructura arborecente permite llegar directamente a través de los descriptores a los sitios de interés o sea a las intersecciones entre polígonos: las cuales son calculadas en el lugar donde existen, eliminándose una gran búsqueda innecesaria.

Como se ve en el árbol ya no existen matrices o mapas numéricos, solamente listas que indican el camino hacia las intersecciones de áreas de interés, en consecuencia se usa poca memoria y se tiene un rápido acceso.

CONSULTAS

En esta parte se describe un ejemplo de consulta al sistema, el ejemplo propuesto, llamado ejemplo SICA es aclaratorio de la rapidéz del sistema y de los conceptos definidos anteriormente.

Ejemplo SICA: Donde existen rocas metamórficas (piza - rra (P)) y Luvisol (L)?.

Viendo la figura "Generación del árbol reconfigurable". La raíz 1 si tiene P y L, esto se hace consultando solamente el descriptor de propiedades, en dado caso que no hubiera alguna de estas propiedades, el sistema inmediatamente responde "no" y ya no hay mas búsqueda; posteriormente el sistema procede a buscar en los siguientes cuatro hijos: 11, 12, 13 y 14, 11 y 14 tienen P y L ahora se busca en sus respectivos hijos: 111, 112, 113, 114, 141, 142, 143 y 144; en los hijos de 11 no hay intersección, en 143 tampoco, en 141 si hay intersección y por lo tanto este es el resultado; ahora se usa cualquier método simple de intersección y el resultado se muestra en la "figura Consultas".

Figura. Consultas

CONCLUSIONES Y RECOMENDACIONES

1. El diseño del SICA esta enfocado para el procesamiento de información geográfica abundante y por lo tanto se considera recomendable su utilización, ya que su método de búsqueda permite la eliminación inmediata de zonas improbables.

2. El árbol reconfigurable es definido por el sistema en base a los datos como la mejor estructura del mismo.

3. Este diseño puede ser aplicable a problemas de clasificación en imagenes multiespectrales digitales ya sean de avión o satélite; agilizando el acceso a la información de cualquier método de clasificación.

4. El usuario puede detener en cualquier momento la consulta al sistema, permitiendole resultados a diferente precisión.

REFERENCIAS

1. BRIBIESCA, E. "MANUAL DE UTILIZACION DEL BANCO DE DATOS CETENAL".

   MEXICO 1977.

2. MARTIN, J. "PRINCIPLES OF DATA - BASE MANAGEMENT".

   Prentice - Hall, Inc.
   Englewood Cliffs, New Versey.

   1976.

3. SALMAN, C. "TECNOLOGIA Y MATERIALES DE DETENAL EN EL ESTABLECIMIENTO DE SISTEMAS ESTATALES DE INFORMACION PARA EL DESARROLLO".

   MEXICO 1977.

4. "SYSTEM OF AUTOMATED CARTO - GRAPHY DETENAL".

   DETENAL
   Research Department.

   MEXICO 1978.

centro de educación continua
división de estudios de posgrado
facultad de ingeniería unam

CASOS PRACTICOS EN EL DISEÑO DE SISTEMAS DE INFORMACION

ORGANIZACION DEL GRUPO DE TRABAJO PARA EL DISEÑO Y LA PROGRAMACION

PROGRAMACION ESTRUCTURADA

-DOCUMENTACION-

Dr. Marcial Portilla Robertson

Septiembre, 1980

CASOS PRACTICOS EN EL DISEÑO DE SISTEMAS DE INFORMACION.

1.- ORGANIZACION DEL GRUPO DE TRABAJO PARA EL DISEÑO Y LA PROGRAMACION.

2.- PROGRAMACION ESTRUCTURADA
- DOCUMENTACION.

ORGANIZACION DE LAS FACILIDADES DE COMPUTO.

- DONDE VAMOS A PONER LOS DATOS ?
- COMO VAMOS A ALMACENAR LOS DATOS ?

[illegible] FACILIDAD DE ACCESO Y ORGANIZACION DE LA BASE.

- SECUENCIAL
- COMO PLUMAS.

ORGANIZACION DE PROCESO

- MAQUINAS DE PROGRAMACION.

una organización (base de datos) tendría algo de la siguiente (ver fig 1).

Administrador de datos.

Las atribuciones y cargos del Administrador de la base de datos y de su staff.

1.- Actúa como un director de la empresa y de la base de datos, así como de un consultor de la organización.

2.- Diseñar un sistema de seguridad en la base de datos.

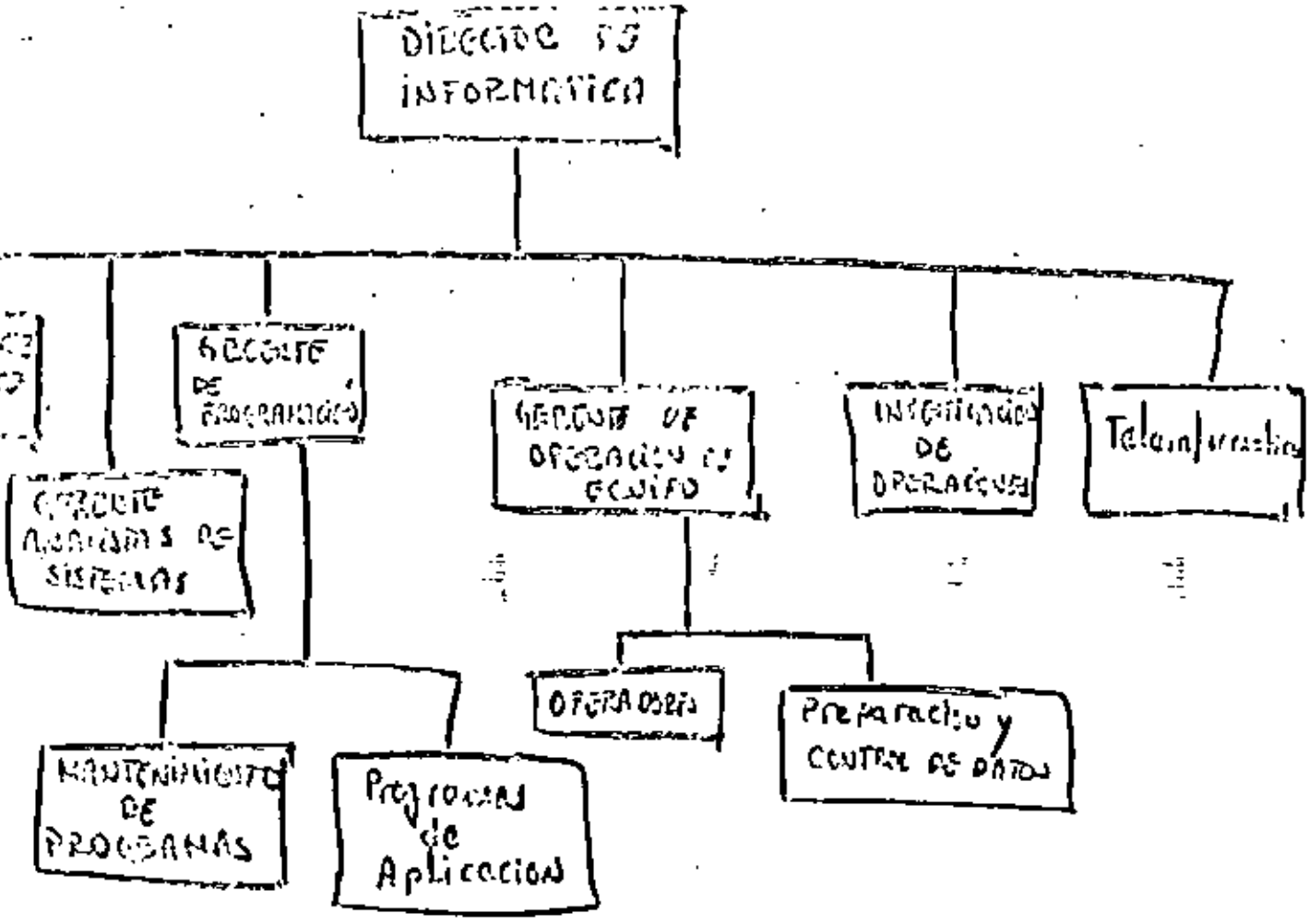


Fig 1.

Auditoría de sistemas — Área con más incidencia
[illegible] y la organización EDP

Funciones de Programación

- [illegible]
- [illegible]
- Análisis sistemas
- Programación

Operación del equipo

Personal en el [illegible] y Proceso interno

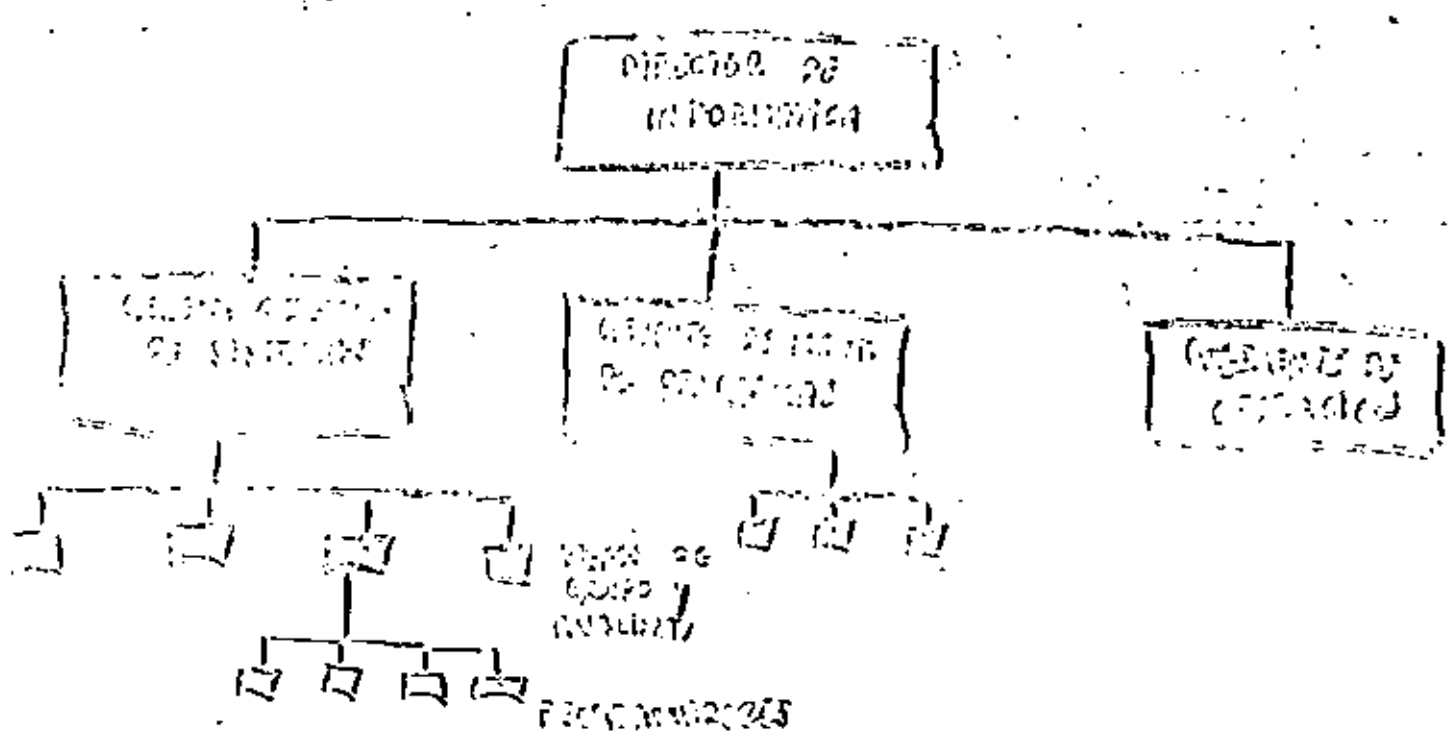
- Tareas
- Responsable [illegible]
- Responsable de los trabajos
  - Hojas de trabajos
  - Cinta de trabajos.
  - Cartas.
- Mantenimiento.

Interpretación de instrucciones — envía los [illegible] de la computadora, así como de la memoria

- Circuitos.
- Soporte lógico.

Telecomunicaciones

- Redes.
- Protocolos
- Seguridad
- Equipo.
- Etc.



EL STAFF (LAS PERSONAS)

LAS PERSONAS EN EL GRUPO NOS INTERESAN EN LA CONDICIÓN DE CANDIDATO (y empleado posteriormente).

- SELECCIÓN DE PERSONAL.
- ENTRENAMIENTO DEL PERSONAL.
- ACTUALIZACIÓN DEL PERSONAL.
- MOTIVACIÓN DEL PERSONAL.

ANTES DE SELECCIONAR AL PERSONAL SE DEBE HACER UNA DE
(DE OBJETIVOS) DEL TRABAJO QUE DEBE DESARROLLAR.

- QUE TRABAJO VA A DESARROLLAR.
- OBLIGACIONES
- EQUIPO NECESARIO.

CONOCIMIENTOS DE CONOCIMIENTO
- EXPERIENCIA
- ENTRENAMIENTO (CONOCIMIENTO)?
- APTITUDES FÍSICAS
- PERSONALIDAD
- ACTITUDES HACIA (O RECIBIDAS).

PERFIL GENERAL DEL PERSONAL.

GERENTES DE INFORMÁTICA — este llevar a cabo funciones de planeación, organización, selección y control. Para llevar a cabo estas funciones los gerentes deben poseer conocimientos especializados en áreas técnicas, habilidades gerenciales en el manejo de personal.

- NO todas las técnicas fueron adquiridas académicamente y NO todas las técnicas académicas son útiles.

- El gerente debe entender la organización de la empresa así como sus metas, y los procedimientos en el manejo de datos.
  - GRADO UNIVERSITARIO
  - CURSOS ADMINISTRACIÓN
  - ECONOMÍA.
  - ESTADÍSTICA.

ANALISTA DE SISTEMAS. (actual, Senior, Junior) ©

- Es quien diseña el sistema propuesto para resolver y atiende los requerimientos y normas básicas.

- Determina los requisitos de la información

- Planea, organiza e integra los componentes existentes en un (nuevo) sistema que resuelva esto.

PROGRAMADOR. — tiene la interpretación del análisis de sistemas y la especificación de requerimientos.

El programador requiere:

- Lenguaje de programación específico
- Utilización de herramientas actuales.
- Crear programas sin una dependencia total
- Policía (en la codificación de errores)
- Creatividad.

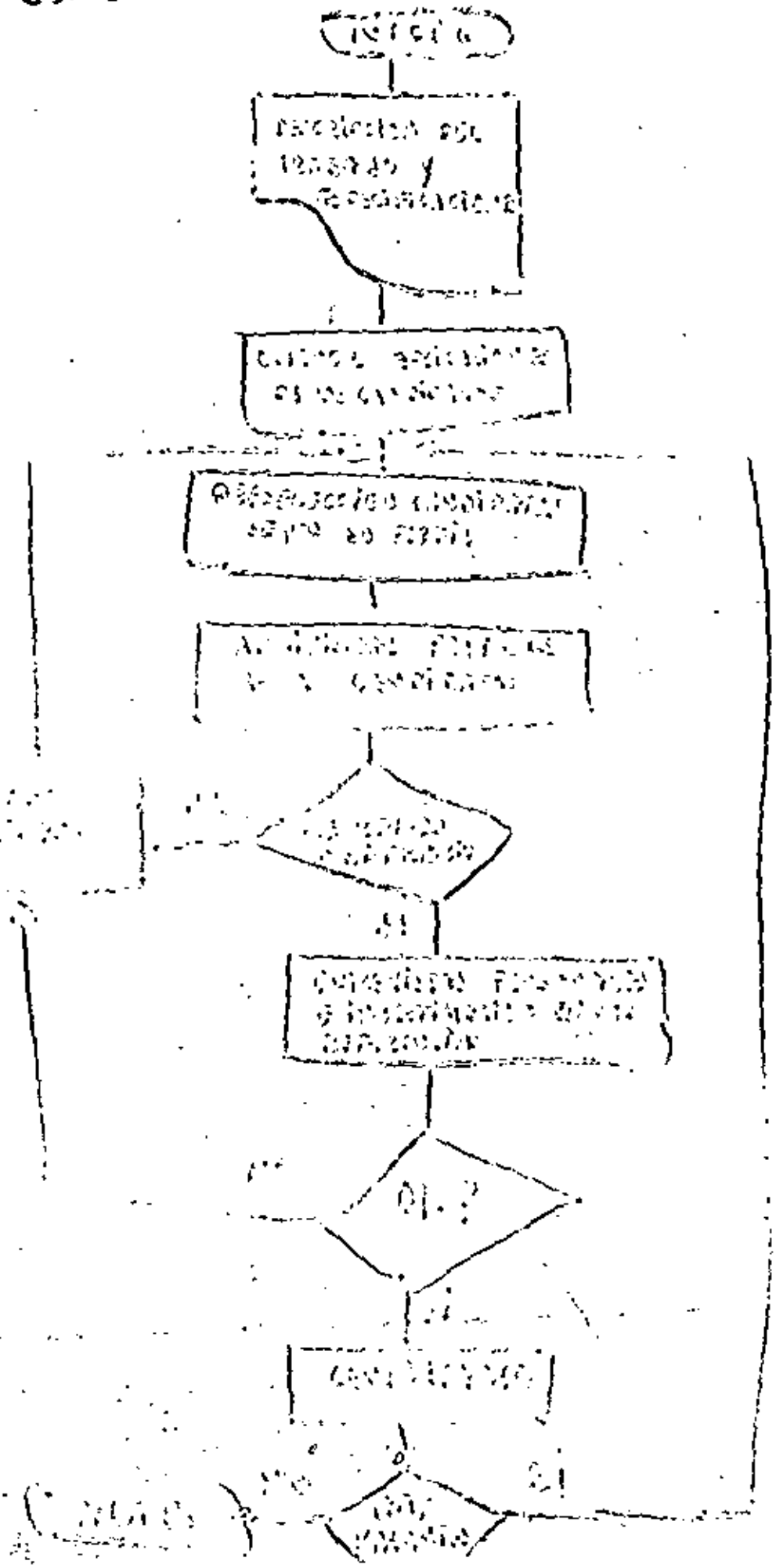
ADMINISTRADOR DE LA BASE DE DATOS. El administrador de base de datos (D.B.) debe:

- Monitorear y llevar acabo auditorías en las operaciones de la base de datos, con el fin de mantener su eficiencia
- Interactuar con los usuarios.
- Coordinar con los usuarios.

OPERACIÓN DE LA COMPUTADORA.

- Carga los trabajos
- Opera el equipo
- Lleva bitácora de trabajo.

COMO SELECIONAMOS CANDIDATOS



ENTRENAMIENTO DEL PERSONAL ADMINISTRATIVO.

- ENSEÑANZA CONTINUA (C.B.T.)
- EL TECNICO
- UNIVERSIDADES Y ESCUELAS TECNICAS
- CURSOS T.A.T.

MOTIVACION DEL PERSONAL.

- NECESIDADES FISIOLOGICAS.
  - COMIDA, VESTIDO, CASA ETC.
- SEGURIDAD.
- NECESIDADES SOCIALES.
- ESTIMO
- PERTENENCIA.

MEDIDA DE EFICIENCIA.

(COMPUTER, PERFORMANCE, EVALUATION) C.P.E.

- MONITORES DE HARDWARE.
  - USO DE LOS SISTEMAS
  - MEDIOS DE BASURA.
- MONITORES DE SOFTWARE.
  - ESTADO DEL SISTEMA
  - PRODUCCION DE REPORTES
  - UTILIZACION DE RECURSOS

ENTRENAMIENTO DEL PERSONAL SELECCIONADO.

- ENSEÑANZA CLÍNICA (E.E.C.)
- EL PRODUCTO
- UNIVERSIDADES Y CENTROS TECNICOS.
- CETA T.A.T.

MOTIVACIÓN DEL PERSONAL.

- NECESIDADES FISIOLÓGICAS.
  - COMER, BEBER, CASA ETC.
- SEGURIDAD.
- NECESIDADES SOCIALES.
- ESTIMO
- PERSONALES.

MEDIDA DE EFICIENCIA.

(COMPUTER, PERFORMANCE, EVALUATION) C.P.E.

- MONITORES DE HARDWARE.
  - USO DE LOS SISTEMAS
  - CICLOS DE BASURA.
- MONITORES DE SOFTWARE.
  - ESTADO DEL SISTEMA
  - PRODUCCIÓN DE REPORTES
  - UTILIZACIÓN DE RECURSOS

- LOS ACCOUNTING PRINCIPLES.
  - USO DE CPU, CANALES, MEMORIA
  - REGISTROS.
  - ESTIMACIONES, CONTEOS CO-CORRIDOS.

(EL INICIO DE LA ADMINISTRACIÓN DE) LA PLANEACIÓN Y LA ESTRUCTURA ORGANIZACIONAL

- ACTIVIDADES ADMINISTRATIVAS.
  - PLANEACIÓN
    - CORTO Y LARGO PLAZO
    - TOMA DE DECISIONES.

PARA PLANEAR HAY QUE:

- IDENTIFICAR LA PLANEACIÓN
- OBTENER Y ANALIZAR LA INFORMACIÓN SIGNIFICATIVA
- CONSIDERAR SUPOSICIONES.
- EVALUAR Y SELECCIONAR LA ALTERNATIVA MÁS APROPIADA.
- TOMAR DECISIONES.

ORGANIZACIÓN

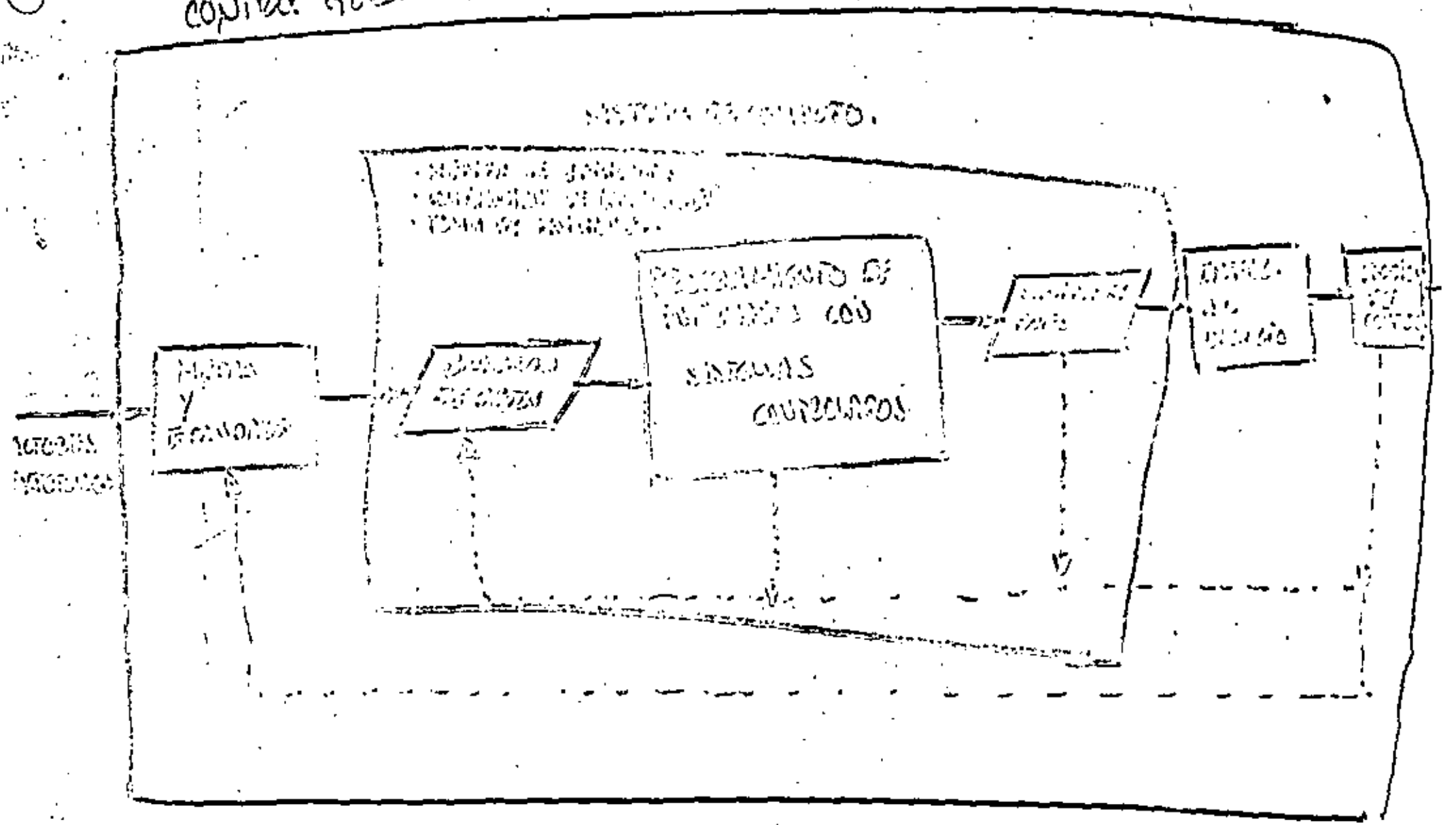
- LA FUNCIÓN DE ORGANIZACIÓN INCLUYE LA SELECCIÓN Y ENTRENAMIENTO DE EQUIPO DE TRABAJO CON EL FIN DE LLEVAR A CABO LOS PLANES

CONTROL

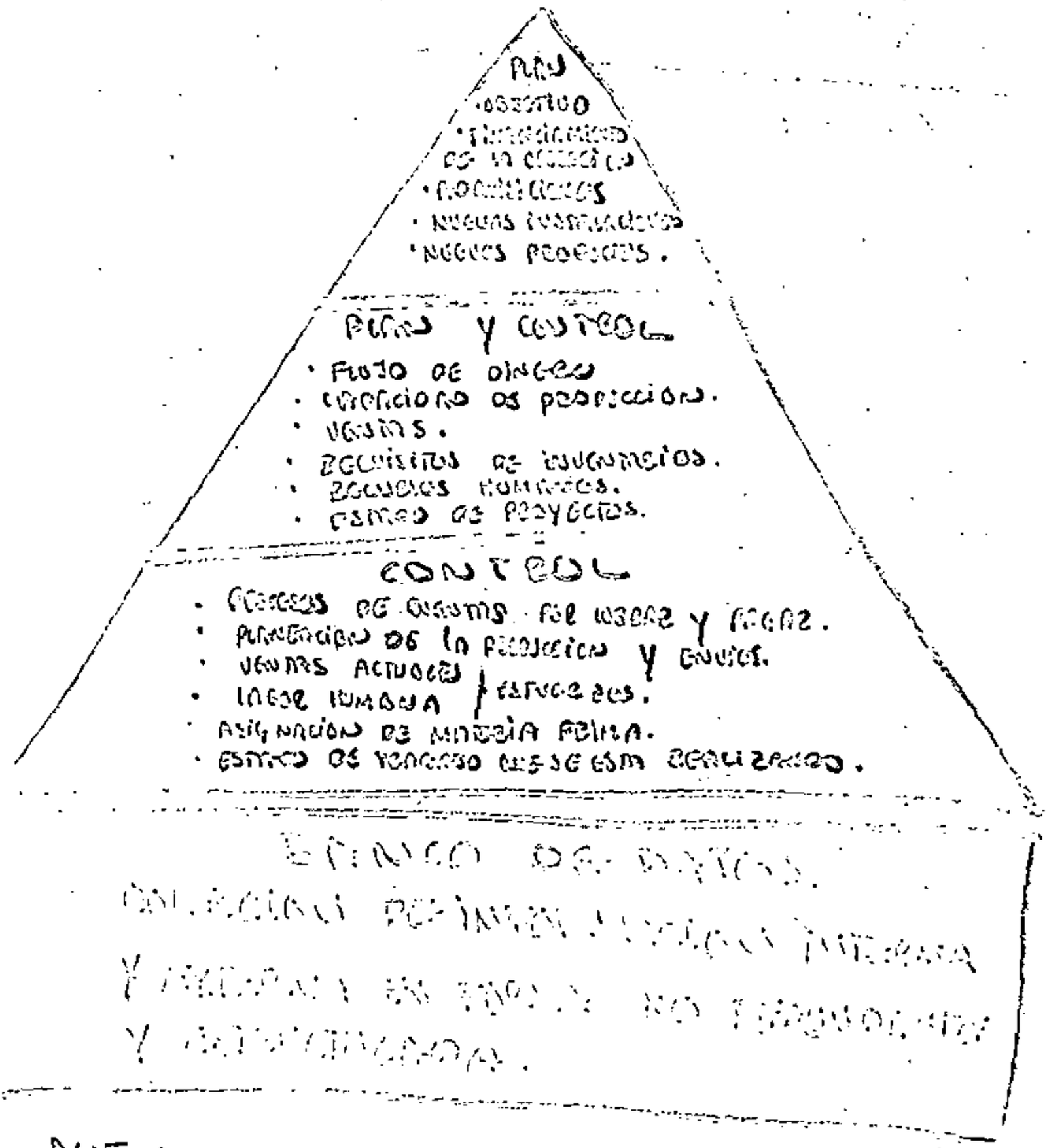
- FIJAR ESTÁNDARES
- MEDIR EFICIENCIA
- COMPARAR LA EFICIENCIA CON LOS ESTÁNDARES
- TOMAR MEDIDAS DE CONTROL

(10)

CONTROL GEOMÉTRICO



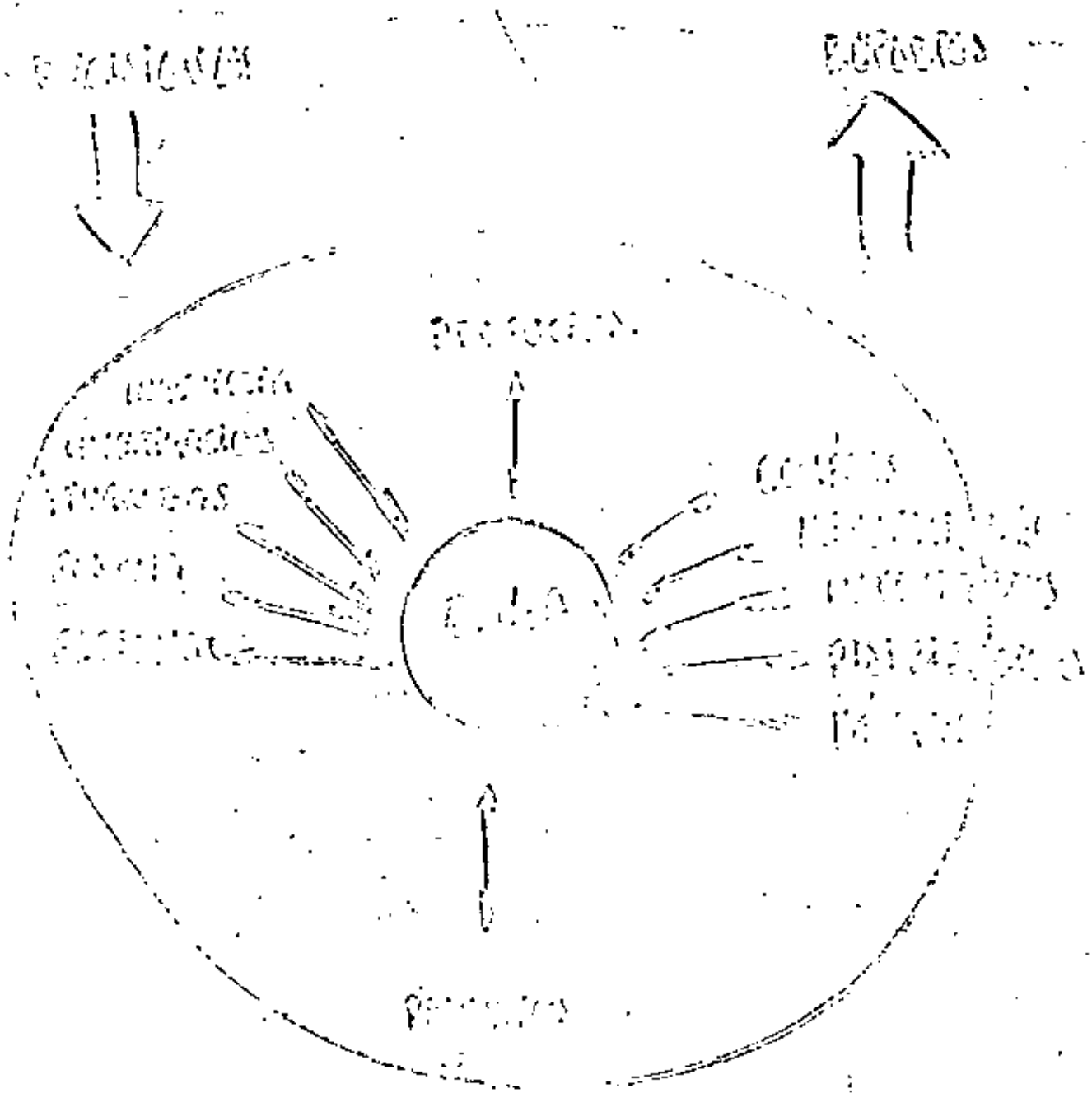
...ORGANIZACIÓN DE EMPRESAS
...CON LA GERENCIA.



ESTADO DE RESULTADOS.
CONTABILIDAD PERMANENTE INTEGRADA
Y PRODUCCIÓN EN PROCESO NO TERMINADOS
Y ACTIVIDADES.

QUE DEMONIOS ES ESO DE "ESTRUCTURADA"

FLUJO DE INFORMACIÓN DENTRO DE LA EMPRESA.
(UNO DE TANTOS)



USO DE LA INFORMACIÓN

- FUNCIONES
- ORGANIZACIÓN
- CONTROL
- PLANEACIÓN

centro de educación continua
división de estudios de posgrado
facultad de ingeniería unam

CASOS PRACTICOS EN EL DISEÑO DE SISTEMAS DE INFORMACION

RECONFIGURACION EN LOS SISTEMAS DE INFORMACION

1. INTRODUCCION
2. EJEMPLO
   a) INTRODUCCION AL EJEMPLO
   b) METODO DE RECONFIGURACION EN EL EJEMPLO
3. CONCLUSIONES

VICTOR GERMAN SANCHEZ

AGOSTO, 1980

RECONFIGURACION EN LOS SISTEMAS DE INFORMACION

1. INTRODUCCION
2. EJEMPLO
   1. INTRODUCCION AL EJEMPLO
   2. METODO DE RECONFIGURACION EN EL EJEMPLO
3. CONCLUSIONES

Victor Germán Sánchez

INTRODUCCION

La eficiencia de un sistema de información en cuanto a su utilización y recursos de computación utilizados puede resumirse en dos factores, tiempo de respuesta y memoria utilizada; donde el primer factor implica por un lado, tiempos cortos de uso de procesador (que se traduce en costo) y por otro, y sobre todo en sistemas de tiempo compartido, una mejor respuesta al usuario; el segundo factor implica menos recursos de memoria utilizados por el sistema (que también se traduce en costo). De esta manera la eficiencia está en la utilización mínima de recursos, procesador y memoria, (costo mínimo) y en un tiempo de respuesta satisfactorio para el usuario.

Por otro lado, debido a que los factores de tiempo y memoria generalmente son inversamente proporcionales (respuesta rápida más memoria, menos memoria respuesta lenta) se plantea el problema de buscar en la implementación de sistemas el punto óptimo en tiempo-memoria a costo mínimo.

Los factores de tiempo y memoria, en cualquier sistema de información, dependen de los algoritmos utilizados en el sistema, de la organización y almacenamiento de la información. Siendo la organización y almacenamiento de la información los que juegan un papel muy importante; dado que la información, generalmente, se almacena en memoria secundaria de acceso directo (por ejemplo disco magnético), el tiempo de acceso a la información es considerablemente mayor al tiempo utilizado para la ejecución de algoritmos.

Así la organización de la información como los métodos de acceso serán los elementos mas importantes a considerar en el diseño e implementación de sistemas de información.

De esta manera para lograr una optimización en tiempo de respuesta y memoria para un determinado sistema de información, lo usual es hacer los programas y la organización de los datos, totalmente orientados a dicho sistema, la desventaja que puede haber en este caso es, cuando cambia la utilización del sistema, pero lo mas seguro es que la organización ya no sea la óptima. Otra alternativa es usar sistemas generales de información, los cuales tienen la ventaja de adaptarse a cualquier cambio en la aplicación del sistema; pero generalmente será difícil asegurar una buena optimización, debido al caracter general del sistema.

De esta manera la reconfiguración se plantea como una solución entre sistemas particulares (óptimos para casos particulares) y sistemas generales (aplicables a cualquier cambio en las aplicaciones).

La idea fundamental de la reconfiguración consiste, en que a partir de una organización inicial, y varias organizaciones posibles y por medio de una información del comportamiento del sistema, en cuanto a su utilización, se hace una evaluación para seleccionar la organización mas adecuada al comportamiento actual y así finalmente reorganizar la información.

A continuación se citará un ejemplo de reconfiguración en un sistema general*.

Primero se describirá brevemente las posibles organizaciones de la información en ese sistema y segundo se verá el metodo de reconfiguración utilizado en dicho sistema (parte del capítulo de reconfiguración).

*"Sistema General de Información Reconfigurable"
Tesis de Maestría, Julio, 1980
Víctor Germán Sánchez Arias.

2. Ejemplo

Introducción

1. Organización Lógica (utilizada en el sistema antes mencionado)

La organización lógica de un archivo consiste, en su representación por una matriz, que tiene como renglones a registros y como columnas a los campos de los registros, los cuales se les llama propiedades. Cada registro tiene al menos una propiedad (común para todos los registros) cuyo valor lo diferencia de todos los demas registros, a esta propiedad se le denominará identificador, y es a través de esta como se pueden consultar cualquier registro del archivo.

En la figura siguiente se muestran los conceptos de archivo, registro, propiedades e identificadores.

PROPIEDADES (COLUMNAS)

NOMBRE	EDAD	TELEFONO
JUAN	20	22113
PEDRO	25	21408
MARIA	18	21101
LUIS	19	29971
JULIA	20	21311
⋮	⋮	⋮

Registros (Renglones)

NOMBRE	EDAD	TELEFONO
JUAN	20	22113

Propiedad nombre = identificador

ARCHIVO, REGISTRO, PROPIEDAD E IDENTIFICADOR.

FIGURA 1

2. Organización física

La información de un archivo se almacena en disco de tres formas: los valores de propiedades se almacenan por renglones (estructura R), por columnas, en forma de vectores binarios, (estructura V) y en ambas estructuras (estructura RV)

Por ejemplo tomando la figura anterior y suponiendo que los valores de propiedad nombre y teléfono se almacenan por renglones se tendrá un archivo R como sigue:

NOMBRE TELEFONO

JUAN	22113.
PEDRO	21408
MARIA	21161
LUIS	29971
JULIA	21311

ARCHIVO R

FIGURA 2

la propiedad edad se almacena por vectores binarios (columnas) de tal forma que se generará un archivo V.

PROPIEDAD EDAD	EDAD
20	10100
25	11001
18	10010
19	10011
20	10100

decimal binario

V_1	V_2	V_3	V_4	V_5
1	0	1	0	0
1	1	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0

ARCHIVO V

ARCHIVO V

FIGURA 3

De manera simple se puede decir que cuando se accesa un registro físico de R se accesan renglones del archivo lógico y cuando se accesa un registro físico del archivo V se accesan columnas binarias de todo el archivo (vectores binarios).

Por último para poder accesar un registro lógico a través de su identificador, se hace por medio de una tabla que contiene el identificador lógico asociado con la dirección física donde se encuentra almacenado, de esta manera preguntar por Juan toma dos accesos, uno a la tabla índice y otro al archivo R.

3. Características de la estructura R.

La estructura R por su tabla índice resulta adecuada para operaciones que involucran a registros particulares, accesándolo por su llave. Una operación típica es la actualización del valor de una propiedad, para lo cual se requiere el identificador del registro, la propiedad a cambiar y el nuevo valor. De esta manera se puede decir que esta estructura esta orientada a las actualizaciones, y en general a las operaciones que accesan por identificador.

4. Características de la estructura V.

Existen otro tipo de operaciones, donde el dato de entrada no es el identificador, si no al contrario, se desea saber quienes cumplen con ciertos valores de propiedades (que puede ser una función lógica, de comparación y aritmética entre propiedades, lo que se denominará predicado). Por ejemplo saber quienes son mayores de 20 años y son mujeres o tienen 18 años y

y son hombres.

Para este tipo de operación, que llamaremos consulta global, la estructura V resulta apropiada, dado que cualquier predicado se puede descomponer en una función logica de los vectores binarios que componen las propiedades involucradas en dicho predicado*.

Para ilustrar un poco esta idea darémos un ejemplo muy sencillo, usando el ejemplo anterior, supóngase que se desea saber quienes tienen 20 años, 20 en binario se representa asf 10100 (usando 5 dfgitos D_1, D_2, D_3, D_4, D_5) de tal manera saber si un número cualquiera es 20 con 5 dfgitos binarios tiene que cumplir la siguiente función logica $D_1 \cdot \bar{D}_2 \cdot \bar{D}_3 \cdot \bar{D}_4 \cdot \bar{D}_5$ (donde "·" = y logico y "¯"=negación), de esta forma, en lugar de dfgitos tenemos vectores binarios ($\underline{V}_1$, $\underline{V}_2$, $\underline{V}_3$, $\underline{V}_4$, $\underline{V}_5$) de tal manera que los que tengan 20 años (deben cumplir la siguiente función logica vectorial $\underline{V}_1 \cdot \bar{\underline{V}}_2 \cdot \bar{\underline{V}}_3 \cdot \bar{\underline{V}}_4 \cdot \bar{\underline{V}}_5$, que aplicada a nuestroejemplo tenemos:

$\underline{V}_1$	$\bar{\underline{V}}_2$	$\bar{\underline{V}}_3$	$\bar{\underline{V}}_4$	$\bar{\underline{V}}_5$	$\underline{V}_{ac}$
1	1	1	1	1	1
1	0	0	1	0	0
1	0	0	0	1	0
1	0	0	0	0	0
1	0	1	1	1	1

$\underline{V}_{ac}$ vector de acceso

OPERACIONES BINARIAS

FIGURA 4

V_{ac} es el vector resultado denominado vector de acceso, donde la posición

* Capítulo OPERACIONES SOBRE LA BASE. Tesis antes mencionada.

de sus "1", indica quienes cumplen con el predicado edad = 20.

La ventaja de esta estructura esta en las operaciones de consulta global, ya que saber quienes cumplen basta tan solo accesar los vectores que estan involucrados en el predicado.

5. Estructura RV.

Sin embargo la estructura R resulta inadecuada para consultas globales, pues se tiene que leer todo el archivo (en V solo se leen algunos registros físicos, vectores binarios). Por otro lado la estructura V no resulta muy adecuada para las operaciones tipo actualizaciones, en cambio en un valor de propiedad requiere del acceso de todos los vectores binarios que componen la propiedad, en R solo se requieren de dos accesos (uno a la tabla índice y otro al registro).

Por esta razón también existirán propiedades en ambas estructuras RV donde las operaciones son actualizaciones y consultas.

VI RECONFIGURACION

La necesidad de reconfiguración en la base de datos nació ante el hecho de dar al sistema generalidad y operación óptima sobre sus operaciones. En el capítulo de Estructura Física, se muestra que para el tipo de operación a la que se orienta a la base habrá implementaciones físicas mejores que otras, además aunado este problema al de no poder saber con presisión cual es el comportamiento de un sistema en cuanto al flujo y operación de datos la posibilidad de un sistema con una estructura óptima invariable (estática) no sería muy real. De esta manera se hace necesario una configuración dinámica que se adapte al tipo de aplicación del sistema y a sus cambios de operación en el futuro.

VI.1 EL PROBLEMA EN GENERAL

La organización óptima, en cuanto a memoria utilizada y tiempo de respuesta, de una base de datos depende de la aplicación (tipo de operaciones mas comunes sobre la base) a la que se oriente al sistema. Esto significa que no existe una estructura única y óptima para un sistema general.

En el momento de creación de una base de datos, se puede conocer de alguna manera (implícitamente en la definición de la base o por información del usuario que la genera) el principal tipo de aplicación que tendrá la base de tal forma que esta podrá ser configurada con una estructura óptima de acuerdo a dicha aplicación.

Sin embargo, si la aplicación no es una característica estática del sistema si no cambia a traves del tiempo, o resulta que la optimización en el momento de creación, un tiempo después, al cambiar la base por los movimientos de datos que en ella se sucedan la estructura ya no sera óptima por no ser la mas adecuada. Por ejemplo, propiedades que eran muy actualizadas son ahora muy consultadas, propiedades que eran muy poco comunes en registro, ahora lo son, etc.

Por lo tanto un sistema constantemente óptimo requiere de una reconfiguración continua.

VI.2 TIPOS DE RECONFIGURACION

La reconfiguración la dividimos en dos tipos:

La que no modifica la estructura de la base, la información solo se reorganiza.

La que modifica la estructura de la base de datos.

En el primer tipo la aplicación (uso de los datos) no ha cambiado, sin embargo la base de datos si ha cambiado, debido a las continuas actualizaciones de los datos, como son las altas, bajas y modificaciones de registros y/o propiedades y de los archivos. De tal forma que la base para conservar un manejo óptimo tendrá que asimilar o actualizar cierta información.

criterio de selección decidir o no el cambio a una estructura distinta en base a un óptimo aprovechamiento de tiempo (mínimo de respuesta) y memoria (mínima memoria utilizada).

Para poder evaluar una estructura se tendrá que medir el movimiento de los datos que en ella se suceden; de esta manera lo que define el movimiento de los datos es la frecuencia de operaciones que se realizan sobre registros y propiedades. Por ejemplo, definiendo dos operaciones, consulta global y actualización, un tiempo después de estar usando la base habrá propiedades muy actualizadas, registros muy consultados, etc. de esta forma se definirá la frecuencia de operación como el número de veces que se ha realizado cierto tipo de operación en un lapso de tiempo dado. Por ejemplo, una frecuencia de operación igual a 8 para la operación de consulta sobre la propiedad edad, significa que hasta ese momento la edad ha sido consultada 8 veces.

De esta manera en forma general, cada elemento del archivo renglón o columna, de acuerdo al tipo de estructura implementada deberá contener dicha información estadística, frecuencia de operación.

Este tipo de reconfiguración depende de la estructura y organización en particular de la base y las funciones a realizar serán tales como, integración de datos en áreas de desbordamiento a los archivos, reordenamientos de archivos, mezclas de archivos, etc.

De esta forma en este tipo de reconfiguración los datos solo se actualizan y se reorganizan sin modificar su estructura básica.

En el segundo tipo de reconfiguración, la aplicación (tipos de operaciones) ha cambiado en algunos archivos por lo que se requiere una restructuración que optimize la nueva aplicación.

Para este tipo de reconfiguración se requiere que al menos la base, sea capaz de soportar dos tipos de estructuras diferentes. Por ejemplo para una organización de datos matricial se puede estructurar para dos aplicaciones, por columnas para consultas globales y por renglones para actualizaciones.

VI.3 METODO DE RECONFIGURACION, FRECUENCIA DE OPERACION Y COSTO DE RECONFIGURACION

La reconfiguración consistirá en la evaluación de las diferentes estructuras de la base y de la selección de la mas adecuada. La evaluación permitira conocer el comportamiento de la base a traves del tiempo, lo cual reflejará el tipo de movimiento de los datos, por las diferentes operaciones que se realicen dada una cierta estructura; para posteriormente con un

	FCC	5	8	3	5
	FCA	4	2	1	4
FRC	FRA				
10	1	JUAN	23	M	3
2	5	MARIA	25	F	5
8	3	PEDRO	24	M	3

FRECUENCIA DE OPERACION SOBRE PROPIEDADES

ARCHIVO LOGICO

FRECUENCIAS DE NOM ED SX CRR

Operación sobre registros:

NOM - Nombre

ED - EDAD

SX - SEXO

FCC - FRECUENCIA SOBRE COLUMNA Y CONSULTA

FCA - FRECUENCIA SOBRE COLUMNA Y ACTUALIZACION

FRC - FRECUENCIA SOBRE RENGLON Y CONSULTA

CRR - CARRERA

FRA - FRECUENCIA SOBRE RENGLON Y ACTUALIZACION

FRECUENCIAS DE OPERACION

FIG. VI.3.1

En esta figura se ilustran las frecuencias de operación para una organización matricial que en el caso mas general puede haber hasta cuatro tipos diferentes de frecuencias de operación para dos operaciones, consulta y actualización. Frecuencia de operación sobre registros de consulta y actualización (FRC y FRA) y frecuencia de operación sobre columnas de consulta y actualización (FCC, FCA). Por ejemplo, el registro María ha sido consultado dos veces y actualizado cinco hasta ese momento. La propiedad edad ha sido consultada ocho veces y actualizadas dos.

El número de frecuencias utilizadas en una representación depende de las estructuras con que se cuente y del número de operaciones que se consideren.

El criterio de selección (costo de cambio). El criterio de selección consiste, dada una cierta estructura y el conocimiento del comportamiento de la base, atraves de sus frecuencias de operación, en evaluar la estructura actual y las posibles para así decidir por la más óptima en memoria y tiempo de respuesta. De esta forma el criterio consistirá en cuantificar en términos de memoria y tiempo de respuesta (costo de cambio) lo que le cuesta al sistema responder a las operaciones con la estructura actual así como cuanto le costaría con cada una de las otras posibles estructuras y decidir por la del costo mas bajo.

Cabe aclarar que se parte del supuesto de que el sistema tendrá un comportamiento, en un lapso de tiempo, igual al comportamiento que se midió en ese momento, (a menos de usar una función predictora).

El costo de cambio tendrá que considerar dos costos elementales, el costo de operación y el costo de reconfiguración. El primero es lo que significa en memoria y tiempo de respuesta a las diferentes operaciones para las diferentes estructuras y el segundo, es lo que cuesta en memoria y en tiempo llevar a cabo físicamente la reconfiguración.

Por ejemplo, supóngase que una propiedad de un archivo, por razones de operación justifique un cambio de estructura, pero sin embargo se de el caso que el costo de reconfiguración física resulte muy alto, debido quizás a que el archivo es muy grande; así en términos totales no es aún justificable un cambio, pues la solución resulta mas cara que el problema.

OPERACIONES	RENGLONES	COLUMNAS	
ACTUALIZACIONES	20	30	COSTO DE OPERACION
	0	15	COSTO DE RECONFIGURACION
CONSULTAS	18	5	COSTO DE OPERACION
	0	10	COSTO DE RECONFIGURACION
	38	60	COSTO TOTAL

FIGURA VI.3.2

En esta figura se ilustra los costos en tiempo para las operaciones de consulta y actualización para una base de datos, con

dos diferentes tipos de estructura (renglones y columnas), dada una estructura actual por renglones. En este ejemplo se ve que, aunque por consultas convendría una estructura por columnas, sin embargo desde un punto de vista global la organización por renglones resulta aun adecuada. Deberá existir una tabla similar para los costos en memoria utilizada.

VI.4 RECONFIGURACION AUTOMATICA

Los cambios de estructura de una base de datos puede ser realizada desde manualmente, por decisión del usuario o hasta totalmente automático y transparente al usuario. La primera es muy simple pues no se requiere de un programa de decisión para reconfiguración; el trabajo lo realizará el usuario, lo que implica el conocimiento de las eficiencias de las estructuras utilizadas y del comportamiento de la base, lo cual generalmente será imprácticó, a menos que el usuario sea muy experimentado y lleve el mismo las estadisticas de uso del sistema. El segundo metodo aunque un poco mas complicado evita al usuario preocuparse por estructuras.

Existe un punto intermedio el cual puede ser interesante y que puede ser opcional cuando se tiene un sistema automático, es aquel sistema que acepte sugerencias u ordenes por parte del usuario para cambios de estructura, es decir un sistema semiautomático. Por ejemplo puede suceder que un sistema tendrá un cambio radical en cuanto a su aplicación la cual se conoce, por lo que podría forzar en un momento a la reconfiguración sin esperar que las frecuencias de operación lo reflejen un tiempo des-

pues. Esta información por parte del usuario podra ser, no necesariamente decisiva pero si importante para decidir una reconfiguración.

VI.5 PROBLEMAS DE IMPLEMENTACION EN LA RECONFIGURACION

El problema de la reconfiguración hasta aqui expresado, parte sobre el supuesto de una frecuencia de operación por propiedad independientemente de las demas frecuencias de operación de las otras propiedades. Es decir hasta ahorita las frecuencias de operación sobre propiedades solo han contemplado consultas y actualizaciones sobre propiedades aisladas y no en conjunto.

Para ahondar mas sobre el respecto, con el esquema hasta ahora presentado, para N propiedades se llevan 2*N frecuencias de operación (N para consultas y N para actualizaciones), esto resultará adecuado si las operaciones son siempre sobre una propiedad, pero por ejemplo si no solo se hace la pregunta a un archivo de empleados quienes tienen 30 hrs. de trabajo (lo que afectaría a la propiedad horas de trabajo en su frecuencia de operación de consulta independientemente de las demas propiedades, y que de hecho lo es en este caso) se hiciera ademas el siguiente tipo de preguntas, incluyendo mas propiedades, como quienes trabajan 30 hrs. a la semana y son mujeres, con el mismo esquema a cada propiedad se le afectaría sus frecuencias de manera independiente entre ellas. En este caso el esquema no es válido por que no es lo mismo esa pregunta si en un caso la estructura para la propiedad hrs esta en columnas y la propiedad sexo esta en vectores y

en otro caso ambas propiedades estan en vectores, puesto que si ambas propiedades estan en vectores el número de accesos es igual al número de bits que componen ambas propiedades, pero si una propiedad esta en R de nada sirve que la primera este en V pues ya no toman el mismo número de accesos cuando ambas estan en V, de esta manera las frecuencias de operación dependen en ciertos momentos de las demás frecuencias de operación cuando hay una operación que involucra a varias propiedades.

Por esta razón para poder llevar un comportamiento total de cada una de las propiedades en un archivo habrá que considerar de manera general que las operaciones tienen que considerar el número de posibles combinaciones que se puedan hacer con todas las propiedades, es decir se requieren de un número de frecuencias igual a 2 * (2 EXP N) donde N es el número de propiedades en el archivo. Si N=10 se requerirán de 1024 frecuencias de operación, que como se observa resultaría muy costoso llevar la estadistica total.

De esta forma, dado el alto costo que tendría que pagarse por una estadistica todal sobre todas las posibilidades (no necesariamente usadas) en frecuencias de operación, la solución tendrá que tomar otro camino un tanto intuitivo para abatir dicho costo. Las soluciones a este caso son variadas. Una primera consideración a tomar será que las estructuras no serán excluyentes, es decir, además de una estructura por renglones y de la de columnas se sumará la de renglones y columnas.

En términos generales ha sido planteado el problema de la reconfiguración en el siguiente inciso se tratará el tema mas concretamente sobre la estructura que se implemento.

Descripción breve de implementación de la reconfiguración, asum.

1. INFORMACION DE COMPORTAMIENTO

Se lleva una estadística de comportamiento de utilización por propiedad Nº de consultas Nº de actualizaciones, Nº de consultas RV estando en V, de esta manera el descriptor de un archivo tiene para cada propiedad:

PROPIEDAD X

NC	NA	NCRV	E

Descriptor de archivo

NC - Nº de consultas

NA - Nº de actualizaciones

NCRE - Nº de consultas RV en tanto la propiedad en V

E - Estructura de la propiedad (R, V, o RV)

2. ORGANIZACION INICIAL

La organización inicial de cada una de las propiedades se lleva a cabo por la información que se da en la creación de la base, sobre la naturaleza de las propiedades, las cuales en cuanto a descripción breve de la implementación de la reconfiguración utilizada

los posibles valores de una propiedad pueden ser definidos (por ejemplo color de coche, podrán existir los colores) o indefinidos (nombres, que pueden existir en una cantidad ilimitada), en cuanto al tipo serán numéricos a alfanuméricos, esta forma, inicialmente toda propiedad definida (numérica o no) se implementará en V (por que existe la posibilidad de preguntas de quienes cumplen para un cierto valor, por ejemplo, dame los que tienen coche verde) y las propiedades indefinidas se estructuran en R (las operaciones mas factibles es el acceso a registros particulares, por ejemplo dame información del empleado X).

Es claro que en la utilización corriente la estructura cambie debido a la reconfiguración.

3. MEDIDA DEL COMPORTAMIENTO

Cada vez que se opera la base de datos se acumulará en la información de comportamiento para cada propiedad involucrada en la operación, así por ejemplo la propiedad edad fue accesada por R se le incrementará NA.

El NCRV se incluye por la razón de que si solo se llevan NC y NA resulta bien cuando cada operación incluye una sola propiedad, pero cuando se incluyen a varias propiedades (que pueden estar en R, V o RV) siempre se hará una solución tipo R (las que están en V se transformarán en R).

De tal manera que puede resultar una consulta y estar la propiedad en V (aparentemente óptimo), sin embargo, la solución no se haya dado en V por que la operación incluye a una propiedad en R.

De esta manera con este factor NCRE se utilizará como un factor de corrección a consultas de propiedades en VC (que en operaciones compuesta o ya no es óptima), de tal forma que influya en un cambio a R. (Para llevar un comportamiento total sería necesario llevar estadísticas sobre todas las posibles combinaciones de propiedades en las operaciones lo cual es del orden 2^N-1 donde N es igual al Nº de propiedades de lo que se observa una estadística total sería muy costosa, para N=10 se requieren 1023 estadísticas).

EVALUACION

En un momento dado se evalua cada archivo de la base de datos, se evalua el costo de operación con las estructuras actuales y el costo con las estructuras propuestas por la información de comportamiento, consultar a V, actualizaciones a R y dando un peso a R para consultar RV, si el costo de la estructura actual es mas bajo, no se reconfigurará (la estructura actual es la mas conveniente), en caso contrario se evalua el costo de cambio, si este es menor o igual que la diferencia de costos entre el costo actual y el costo si estuviera reconfigurado, el archivo se reconfigura.

3. CONCLUSIONES

Si bien la reconfiguración pretende optimizar recursos y tiempos de respuesta, no hay que perder de vista, que la reconfiguración tiene su propio costo, y en un momento dado puede ser muy alto. Por otro lado hay que hacer notar que el análisis en el ejemplo citado esta basado en memorias de acceso directo, como disco; no se toman en cuenta otras alternativas, como lo sería la construcción de un "disco" al que se le pueden accesar renglones y columnas de manera directa, o usar como sistema de almacenamiento a memoria principal (alternativa cada vez mas viable , por el continuo decrecimiento de los precios en las memorias de este tipo).

Sin embargo, tomando en cuenta las consideraciones antes mencionadas la reconfiguración puede ser una solución al problema de optimización de recursos y tiempo de respuesta para los sistemas generales de información.

Directorio de Asistentes al Curso: "Casos Prácticos en el Diseño de Sistemas de Información 1980.

1. José Francisco Arellano Alvear
Instituto de Investigaciones Eléctricas
Leibnitz No. 1
Mexico 5, D.F.
525 20 28

Velázquez de León 52
Col. Sn. Rafael
Z.P.4

2. Joaquín Arellano Núñez

3. Miguel Alejandro Arenas Rodríguez
COCONAL, S.A.
Periférico Sur 6501
Z.P.22
676 55 55 Ext. 116

Relojeros 46
Col. Retoño
Z.P.13
532 94 09

4. Gerónimo Armijo Burciaga
S.A R H
Reforma 107=5°
Z.P.4
592 10 31

Insurgentes Nte. 21-C
Z.P.3

5. Sergio Briseño Sánchez
Cía. de Luz y Fza. del Centro, S.A.

6. J. Guadalupe Roberto Carbajal Solís
I M P
Lázaro Cárdenas 152
Z.P.14
567 66 00 Ext. 2686

Pte. de Calderón 35-17
Sn. B. Atepehuacán

7. Juan Carlos Chu Pulido
S A H O P
Reforma 77-10 °
Z.P. 4
534 25 97

Sn. A. Abad 29-2 A
Z.P. 8
761 13 85

8. Conrado Dávila García
D. D. F.
SN. A.Abad 231-7°
Z.P. 8
588 32 27

Miguel Laurent 61-202
Z.P.12

9. Manuel Horacio De la Rosa Lazos
Fac. de Est. Sup. Cuautitlán Campo 3
Cuautitlán Izcalli, Edo. de Méx.
33 111 Ext. 391

Alhambra 413-4
Col. Portales
Z.P.13
539 99 83

10. José María de la Torre Wolf
S A R H
Jefe del Depto. de Evaluación del Desarrollo
Reforma 107-7°
Z.P.4
535 02 52

Hda. Sn. José Vistahermosa 27
Echegaray
Naucalpan, Edo. de Méx.
373 50 25

11. José de Jesús de los Santos y del Angel
Banco de Méx., S.A.
Insurgentes Sur 2375-5°
Z.P.20
550 70 11 Ext. 168

Laguna de Guzmán 119-5
Z.P.17

12. Enrique Espriella Medina
Cía. de Luz y Fza. del Centro, S.A.
Av. Melchor Ocampo 173 Oficina 402
Z.P. 17
546 67 96

Juan O'Donojú 256
Lomas de Chapultepec
Z.P. 10
540 45 76

13. Jorge Fernández Moreno
C. F. E.
Ródano 14
Z.P. 5
511 00 99

Amores 1874-902
Z.P.12
575 42 76

14. Miguel Flores Ortega
C. F. E.
Melchor Ocampo 469-101
Z.P. 5
511 00 99

11 de Abril 177-503 A
Sn. P. de los Pinos
Z.P.18

15. José Luis Frías Alcaraz
C. F. E.
Ródano 14
Z.P. 5

Sauces 11
Jardines de Sn. Mateo
Naucalpán, Edo. de Méx.
511 00 36

16. Salomón Frid Ran
D. D. F
Ofi. del Sist. de Infor. de Desarrollo Urbano
Pino Súarez 15
Z.P.1
522 64 38

Chego 14-502
Col. Hipódromo
México, D.F.
574 55 66

17 Alfonso Galindo Valencia
C. F. E.
Melchor Ocampo 469 1° Piso 101 y 102
Z.P.5
511 00 99

18. Anatolio García Monfil
Colegio de Bachilleres
Av. Cuauhtémoc 1236
Sta. Cruz Atoyac
México
559 55 22

Rep. de Chile 12 Altos
Centro
Z.P. 1

19. Juan Pablo García Morales

20. Ana María García Melero
PEMEX
Marina Nal. 329
Z.P.17

San Pedro 194
Coyoacán
Z.P. 21
554 47 87

21. Teodoro González Esteban
Inst. de Inv. Eléctricas
Leibnitz 14-9° Desp. 901
Z.P.5
525 62 73

Milwaukee 42
Col. Nápoles
511 68 64

22. José Gustavo Gorocica Palma
SAHOP
Miguel Laurent 840
Z.P. 12
559 16 38

Pilares 1417
Z.P.13
672 48 09

23. Rogelio Fernando Hernández Miranda
Centro de Cómputo
Av. de la Juventud s/n
Chilpancingo, Gro.
227 41

Niños Héroes 15
Chilpancingo, Gro.

24. Manuel Hernández Romero
PEMEX

Pilares 94
Z.P. 12
575 33 77

25. Salvador Hernández Ugalde
Colegio de Bachilleres
Av. Cuauhtémoc 1236
Z.P. 13
559 55 22 Ext. 140

Calle 4 # 10-13
Col. Independencia
Z.P.13
532 23 64

26. Laura Herrejón Caballero
Inst. de Inv. Eléctricas
Leibnitz 14 PH
Z.P. 5
250 53 16

López Cotilla 1544-202
Z.P.12
525 69 79

27. Mario Herrera Carballido
S A R H
Reforma 107-5°
Z.P. 4
592 10 31

Miguel Negrete 20-3
Niños H. de Chap.
México, D.F.
590 60 91

28. Ma. Elena Huerta López
PRODEL, S.A.
Hamburgo 31-1°
Z.P. 5
591 17 27

Dr. Atl 260-16
Sta. Ma. la Ribera
Z.P. 4
547 47 03

29. José Dionicio Izquierdo Buenrostro
COCONAL S.A.
Periférico Sur 6501
Z.P. 22
676 55 55 Ext. 116

Condor 154
Col. Aguilas
Z.P. 21

30. Armando Jasso Arfas
S. A. R. H.
Reforma 107-8°
Z.P. 4
592 10 62

Calle 25 No. 94
S. P. de los Pinos
Z.P. 18
515 80 44

31. Alejandro Jiménez F.

32. Arturo Jiménez Mayén
UAM
Calz. del Hueso s/n
México, D.F.

Av. 531 No. 89
U. Aragón
Z.P.14
551 12 21

33. Enrique Lastra Moreno
PEMEX

Nte. 9 No. 271
Col. Moctezuma
Z.P. 9
762 03 73

34. Sergio López González
Inst. de Inv. Eléctricas
Leibnitz 14-3°
Z.P.5
531 66 90

Av. 1° de Msyo No. 255
Sn. Pedro de los Pinos
Z.P.18
515. 01. 29

35. Florencio Guillermo Manrique de Lara Gtz.
Médanos 44
Las Aguilas
Z.P. 20
593 33 60

36. Ma. Teresa Medina Vite
Univ. Autónoma de Hgo.
Abasolo 600
Pachuca, Hgo.
2 65 35

Tagle 202
Pachuca, Hgo.
2 41 53

37. Roberto Carlos Mendoza Preciado
Inst. de Inv. Eléctricas
Leibnitz 1-8°
Z.P. 5
525 20 28

Calle 23 No. 91 Altos
Pro-Hogar
Z.P.14

38. José Luis Molina Figueroa
PEMEX

Av. Sn. Mateo Nopala
No. 61-29
Jardines de Sn. Mateo
Naucalpán, Edo. de Méx.

39. Gabino Gaspar Monterrosa Reyes
Dir. Gral. de Const. y Ope. Hidráulica
D D F
S. A. Abad 231-7Piso
Z.P. 8
588 32 27

Saratoga 1017-202
Z.P.13
532 46 59

40. Rodolfo Pimentel Camacho
Cía. de Luz
Tlaloc 90-4°
Z.P. 17
546 11 55

Hda. de Santiago 82
Prados del Rosario
Z.P.16
561 71 37

41. Arturo Ponde de León Huerta
Dir. Gral. de Planificación
Pino Suarez 15
Z.P. 1
522 64 38

Fut bol 118-2
Country Club
Z.P. 21
49 41 82

42. Luis Joaquín Poot Ayala
Dir. Gral. de Planificación
D D F
Plaza de la Const. No. 5
Z.P. 1
530 29 95

Dr. Vértiz 757-27
Z.P.12

43. Justo M guel Ramírez Cabrera
Inst. de Inv. Eléctricas
Leibnitz 1-8°
Z.P. 5
525 38 04

Tripoli 310-306
Z.P. 13
672 05 75

44. Rodolfo Antonio Ramírez Rico
S A R H
P. de la Reforma 69-5°
Z.P.1
546 80 70

Av. 20 de Agosto Edif. 1
Paseos Taxqueña
Z.P.21

45. Raymundo Hugo Range Gutiérrez
Fac. de Ing. Apdo. Postal 20-380
UNAM
México 20, D.F.
550 52 15 Ext. 3750

46. Carlos Reyna Ramírez
S A R H
Ofi. de Regionalización y Clasificación
Hidjrológica
Sn. A. Abad No. 32-1:0 ° Piso

Calz. de los Misterios
No. 89-25
Ex-Hipodrómo de P.
Z.P.2

47. Jorge Arturo Rodríguez Córdoba
Es.de Ing
Universidad Autónoma de G uerrero
Av. de la Juventud s/n
Chilpancingo, Gro.

Av. Guerrero 10-B
Chilpancingo, Gro.

48. Juan José Rodríguez Jiménez
S A R H
Reforma 107-2°
Z.P. 4
566 94 92

Av. Américas 16
Col. Moderna
Z.P. 13
590 16 46

49. Francisco Javier Rodríguez Avila
D. D. F.
Sn. A. Abad No. 231-5°
Z.P. 8
588 78 06

Eten 619-1
Col. Lindavista
Z.P. 14
754 43 69

50. Jorge Silva Midences
Bolivar 745-104
Col. Alamos
Z.P. 13
579 77 90

51. Pedro Verján Vargas
Comisión de Aguas del Valle de Méx.
Balderas No. 55
México 1, D.F.
510 02 94

Virginia 164
Col. Nativitas
Z.P. 13
532 27 32

52. Eusebio Velázquez Hernández
S A R H
Reforma 35 Entrepiso
Z.P.1
546 59 28

Azaharez 26
Fracc. Villa de las F.
Coacalco, Edo. de Méx

53. Vicente Villa Durán
SARH
Reforma 107-2°
Z.P. 4
566 94 92

Sur 123 Manz. 2 Lote 10
Juventino Rosas
Z.P.8
657 78 28

54. Alicia del Carmen Villalpando Rentería
Universidad Autónoma de Hidalgo
Abasolo 600
Pachuca, Hgo.
2 65 35

Tiro Tule 109
Real de Minas
Pachuca, Hgo.
2 08 85