DIRECTORIO DE PROFESORES DEL CURSO
DISEÑO DE BASES DE DATOS
DEL 5 DE JUNIO AL 3 DE JULIO DE 1992.

M. EN I. EFRAIN PARDO ORTIZ
DIRECTOR GENERAL
CENTRAL DE SOFTWARE S.A. DE C.V.
REVOLUCION 1134 2o. PISO, SAN JOSE INSURGENTES, DELEG. B. JUAREZ, C.P. 03900, TEL. 680 04 88 Y 680 60 96

ING. SALVADOR PEREZ VIRAMONTES
ASESOR INDEPENDIENTE
TEL 674 59 46

ING. ALVARO CASTIELLO DE LA HIDALGA
DIRECTOR GENERAL
MEGATRON DE MEXICO
ANAXAGORAS No. 1312, C.P. 03650, MEXICO D.F.
TEL. 604 49 43, 688 99 66 (FAX)

ING. ADOLFO MILLAN NAJERA
SECRETARIO DE LA DIVISION DE INGENIERIA ELECTRICA, ELECTRONICA Y EN COMPUTACION
DIVISION DE INGENIERIA UNAM
TEL. 548 99 58, 550 52 15 EXT. 3744

DIRECTORIO DE ALUMNOS DEL CURSO
DISEÑO DE BASES DE DATOS
DEL 5 DE JUNIO AL 3 DE JULIO DE 1992.

1.- DURAN SANCHEZ JUVENTINO
OPERADOR DE SISTEMAS
BODEGA AURRERA, S.A. DE C.V.
CHIMALPOPOCA 65, COL. OBRERA
TEL. 588 91 11 OFNA., 839 81 99 DOM.

2.- MONROY MONTECILLO MARTIN

3.- NAJERA CAMPOS ROSA MARIA
ASESOR POR HONORARIOS
COLMEX
CAMINO AL AJUSCO No. 20, C.P. 01000
TEL. 645 59 55 EXT. 393 OFNA.

4.- NAVA PEREZ GUILLERMO
INGENIERO DE DISEÑO
INSTITUTO MEXICANO DEL PETROLEO
AVE. LAZARO CARDENAS No. 152
TEL. 368 59 11 OFNA., 792 05 43 DOM.

5.- OROZCO LEYVA FRANCISCO
COORDINADOR DE INVENTARIOS
PETROLEOS MEXICANOS
IBSEN No. 43 8o. PISO, POLANCO
TEL. 280 32 27 OFNA., 587 86 53 DOM.

6.- PEREZ ARTEAGA JOSE GUADALUPE
OPERADOR DE PAQUETERIA Y CAPTURISTA
DIRECCION GENERAL DE CONSTRUCCION Y OPERACION HIDRAULICA
AV. VIADUCTO M. ALEMAN No. 507, COL. GRANJAS MEXICO,
DELEG. IZTACALCO, TEL. 657 74 55 EXT. 250 OFNA.

7.- RODRIGUEZ NERI JOSE LUIS
JEFE DE SECCION
DIRECCION GENERAL DE CONSTRUCCION Y OPERACION HIDRAULICA
AV. VIADUCTO RIO DE LA PIEDAD 507-2o PISO, COL. GRANJAS
MEXICO, DELEG. IZTACALCO, TEL. 657 34 39 OFNA.

8.- SANTOS HERNANDEZ ORLANDO
GERENTE DE DESARROLLO
COMTEC METAMAR, S.A. DE C.V.
FILIPINAS No. 519-8, COL. PORTALES, DELEG. B. JUAREZ
TEL. 672 46 56 OFNA.

9.- TORRES ORTIZ MA. ELENA
PROGRAMADOR
DIRECCION GENERAL DE SERVICIOS MEDICOS, UNAM
CIUDAD UNIVERSITARIA, D.F.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
FACULTAD DE INGENIERIA
DIVISION DE EDUCACION CONTINUA
CURSOS ABIERTOS

DISEÑO DE BASE DE DATOS

05 de Junio al 03 Julio de 1992

FECHA	HORARIO	TEMA	PROFESOR
Viernes 05 de Junio	17:00 a 21:00 hrs.	Introducción a los sistemas manejadores de datos	M. en C. Efraín Pardo Ortiz
Sábado 06 de Junio	9:00 a 14:00 hrs.	Introducción a los sistemas manejadores de datos	Ing. Salvador Pérez Viramontes
Viernes 12 de junio	17:00 a 21:00 hrs.	Administración de base de datos (capítulo 2)	Ing. salvador Pérez v. Ing. Eduardo Hernández Ing. Adolfo Millán Najera
Sábado 13 de Junio	9:00 a 14:00 hrs.	Análisis de requerimientos (capítulo 3)	Ing. Alvaro Castiello de la H.
Viernes 19 de Junio	17:00 a 21:00 hrs.	Modelos de datos (capítulo 4)	
Sábado 20 de Junio	9:00 a 14:00 hrs.	Diseño conceptual (capítulo 5)	
Viernes 26 de Junio	17:00 a 21:00 hrs.	Diseño lógico (capítulo 6)	
Sábado 27 de Junio	9:00 a 14:00 hrs.	Temas complementarios (capítulo 10)	
Viernes 03 de Julio	17:00 a 21:00 hrs.	MOdelo físico (capítulo 7 - al 9)	

EVALUACION DEL PERSONAL DOCENTE

CURSO: *DISEÑO DE BASE DE DATOS*

FECHA: *05 DE JUNIO AL 03 DE JULIO DE 1992.*

	CONFERENCISTA	DOMINIO DEL TEMA	EFICIENCIA EN EL USO DE AYUDAS AUDIOVISUALES	MANTENIMIENTO DEL INTERES. (COMUNICACION CON LOS ASISTENTES, AMENIDAD, FACILIDAD DE EXPRESION).	PUNTUALIDAD	
10	*M. EN C. EFRAIN PARDO ORTIZ*					
11	*ING. SALVADOR PEREZ VIRAMONTES*					
12	*ING. EDUARDO HERNANDEZ*					
13	*ING. ADOLFO MILLAN*					
14	*ING. ALVARO CASTIELLO DE LA H.*					
15						
16						
17						
18						
	ESCALA DE EVALUACION : 1 a 10					

EVALUACION DE LA ENSEÑANZA

CURSO: DISEÑO DE BASE DE DATOS
FECHA: 05 DE JUNIO AL 03 DE JULIO DE 1992.

SU EVALUACION SINCERA NOS AYUDARA A MEJORAR LOS PROGRAMAS POSTERIORES QUE DISEÑAREMOS PARA USTED.

TEMA	ORGANIZACION Y DESARROLLO DEL TEMA	GRADO DE PROFUNDIDAD LOGRADO EN EL TEMA	GRADO DE ACTUALIZACION LOGRADO EN EL TEMA	UTILIDAD PRACTICA DEL TEMA	
Introducción a los sistemas manejadores de datos					
Administración de base de datos (capítulo 2)					
Análisis de requerimientos (capítulo 3)					
Modelos de datos (capítulo 4)					
Diseño conceptual (capítulo 5)					
Diseño lógico (capítulo 6)					
Temas complementarios (capítulo 10)					
Modelo físico (capítulo 7 al 9)					
ESCALA DE EVALUACION: 1 a 10					

EVALUACION DEL CURSO

	C O N C E P T O	
1.	APLICACION INMEDIATA DE LOS CONCEPTOS EXPUESTOS	
2.	CLARIDAD CON QUE SE EXPUSIERON LOS TEMAS	
3.	GRADO DE ACTUALIZACION LOGRADO EN EL CURSO	
4.	CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
5.	CONTINUIDAD EN LOS TEMAS DEL CURSO	
6.	CALIDAD DE LAS NOTAS DEL CURSO	
7.	GRADO DE MOTIVACION LOGRADO EN EL CURSO	
	EVALUACION TOTAL	

ESCALA DE EVALUACION: 1 A 10

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

C U R S O S A B I E R T O S

DISEÑO DE BASE DE DATOS

INTRODUCCION A LOS SISTEMAS MANEJADORES DE DATOS

parte 1

M. EN C. EFRAIN PARDO ORTIZ

JUNIO, 1992.

CURSO ABIERTOS
DISEÑO DE BASES DE DATOS
PARTE I

INTRODUCCION A LOS SISTEMAS MANEJADORES DE DATOS

M. EN C. EFRAIN PARDO ORTIZ

mayo, junio, 1991.

PAPEL DE LA INFORMACION EN LA INSTITUCION

"LA INFORMACION ES EL RECURSO QUE NOSPERMITE APROVECHAR MEJOR TODOS NUESTROS RECURSOS"

INTRODUCCION

La información es un recurso de la empresa pero...

- ¿Le estamos dando dicha categoría?
- ¿Sabemos cuanta información y de qué utilidad, es proporcionada por DP?
- ¿Se podrá permanecer ajeno a las nuevas tecnologías de DP?...¿Cuanto Tiempo?
- ¿Es realmente imprescindible un análisis corporativo?
- ¿Esta el énfasis de los sistemas de información en el proceso de toma de decisiones?

Proceso de datos es una inversión...

- ¿Estamos recuperandola?

VISION ESQUEMATICA DE UNA EMPRESA

MISION	OBJETIVOS	ESTRATEGIAS
• PRODUCTO/SERVICIO	• MAYOR UTILIDAD • NUEVOS PRODUCTOS	• INCREMENTO 10% • NUEVA FABRICA

RECURSOS	ORGANIZACION	FUNCIONES
• PRODUCTOS • CLIENTES • DINERO • FACILIDADES	• PRESIDENCIA • VICEPRESIDENCIA • DIRECTORES • GERENTES	• PRODUCCION • MERCADEO • FINAZAS • DISTRIBUCION

FACTORES DE PLANEACION.

- INTERNOS
  - Proyeccion de ventas.
  - Ganancias.
  - Factores de crecimiento.
  - Plan financiero.
  - Penetración en el mercado.

- EXTERNOS
  - Porcentaje de inflación.
  - Costo del dinero.
  - Nueva legislación.
  - Crísis energética.
  - Competencia (nal/int'al)
  - Obsolecencia.

APOYO A LAS VENTAS

ESTRATEGIAS CORPORATIVAS
• MISION • OBJETIVOS • RECURSOS • ESTRATEGIAS • FUNCIONES • RESTRICCIONES

ESTRATEGIAS INFORMATICAS
• OBJETIVOS • RECURSOS • ARQUITECTURA DE INFORMACION • PROCEDIMIENTOS • NIVELES DE SERVICIO A USUARIOS

TRADUCCION

PROBLEMAS ACTUALES

Credibilidad y comunicación entre profesionales de DP y áreas de toma de desiciones.

"No entiendo lo que dice ésta gente. La computadora es un misterio para mí"

"No parecen entender nuestros problemas"

"Les toma mucho tiempo materializar los resultados"

"El grupo de DP es inflexible"

"No puedo lograr involucrarlos"

"No tienen ideas claras"

"No entienden lo que estamos tratando de hacer"

"Esperan demasiado en muy poco tiempo"

"Después de que liberamos un sistema, todo lo que hacen es criticarlo".

OBJETIVOS INSTITUCIONALES

- **Integrar a la organización.**
- **Mejorar el nivel de servicio.**
- **Agilizar el manejo de recursos.**
- **Reducir costos de operación.**
- **Mejorar la productividad del personal.**
- **Impulsar a la organización.**

LOS OBJETIVOS VERDADEROS DE INFORMATICA EN SU EMPRESA:

- **Contribuir a la sobrevivencia y crecimiento.**
- **Apoyar al proceso de toma de decisiones.**
- **Dar información a todos los niveles.**
- **Ayudar al cumplimiento del ciclo:**
  - **Planeación.**
  - **Operación .**
  - **Control.**
- **Ayudar a planear, diseñar e implantar sistemas de información ajustados a estrategias corporativas.**
- **Analizar nuevos medios de servir efectiva y económicamente.**

CONTROL DEL RECURSO DATOS

- Un libro de 300 pag. ~ 10 caracteres.
- Una empresa mediana ~ 10 caracteres.
- 10,000 libros, mejor que los controlemos!
- Requerimientos. Imprescindibles de
  - Libreros, catálogos, procedimientos.
  - Facilidades (mesas, lámparas, carros, visores)
  - Bibliotecarios, administradores, etc.
- Los sistemas DP tradicionales, no controlan eficazmente los datos.
- DBMS controla centralizadamente los datos proporcionandolos, al usuario autorizado, en el momento requerido.

PARAMETROS DE CALIDAD EN LA INFORMACION

LA INFORMACION DEBE SER:

- Clara.
- Consistente.
- Oportuna.
- Confiable.
- Concisa.
- Atractiva.
- Segura.

PARAMETROS DE CALIDAD DE LA INFORMACION

C L A R A:

La información debe ser expresada en los términos y formatos que le sean más útiles al usuario.

CONSISTENTE

Preferentemente tener solo una versión de cada dato, a menos que, por requerimientos de seguridad o validación, sea necesario un actor de redundancia en cuyo caso deberán estar perfectamente definidos, tanto sus objetivos y vigencias como sus elementos de control, sincronías y jerarquías.

OPORTUNA:

La información puede ser extraída o recuperada en el momento que se requiera.

CONFIABLE:

Las fuentes de generación de la información están claramente identificadas, perfectamente definidas sus responsabilidades para con la organización y cuentan con los recursos necesarios para poder cumplir con ellas.

CONCISA:

Poder proporcionar toda la información necesaria pero a su vez solamente la necesaria para satisfacer adecuadamente cada requerimiento.

ATRACTIVA:

ÇContar con la suficiente flexibilidad en la éstructura de su presentación para llegar a ser un verdadero apoyo en la realización de las tareas de sus usuarios.

SEGURA:

Lograr que los mecanismos de captura, organización, almacenamiento y recuperación permitan simultáneamente un acceso directo y total cuando se requiera y una total confidencialidad cuando así se especifique.

EN RESUMEN:

LA INFORMACION EN UNA ORGANIZACION SE TRADUCE EN:

- PRODUCTIVIDAD.
- EFICIENCIA.
- COMPETITIVIDAD.
- EFICACIA
  - (Mejores y más oportunas Decisiones y Acciones).
- MEJOR CONTROL.
- MEJORES BASES PARA EL PROCESO DE PLANEACION.

CONCLUSION

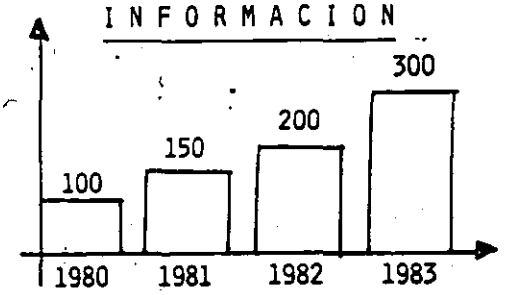
Si la importancia y atención que la organización le presta al recurso información no es la adecuada ¿porqué se espera tener resultados adecuados producto de los procesos de toma de decisiones?

DATOS VS. INFORMACION

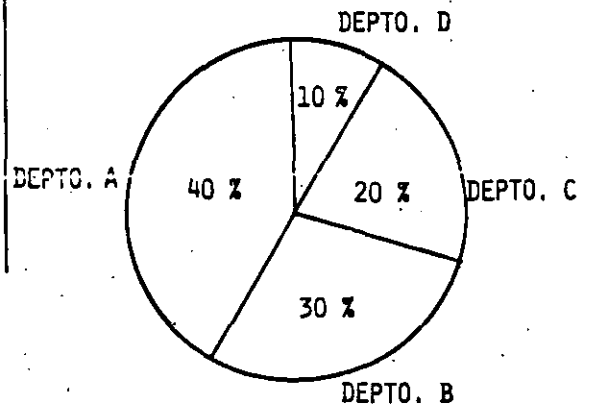
DATOS

1980	100
1981	150
1982	200
1983	300

INFORMACION



DEPARTAMENTO A	10 %
DEPARTAMENTO B	20 %
DEPARTAMENTO C	30 %
DEPARTAMENTO D	40 %



LA MATERIA PRIMA DE LA TOMA DE DESICIONES

ENTRADA	PROCESO	SALIDA
RECURSOS ECONOMICOS	ADQUISICION Y/O CON-TRATCION	BIENES Y SERVICIOS
MATERIAS PRIMAS	PROCESO DE TRANS-FORMACION	PRODUCTOS TER-MINADOS
INFORMACION	ANALISIS Y TOMA DE	DECISIONES

CALIDAD X CALIDAD = CALIDAD

FORMAS DE PROCESO

El procesamiento de los datos puede ser clasificado desde varios puntos de vista:

- **Forma de Ejecución.**
- **Localización.**
- **Grado de Concentración.**

FORMA DE EJECUCION

- **PROCESAMIENTO EN LOTES (BATCH).**
  - La Información a ser procesada se organiza e introduce al Sistema de Procesamiento en "Lotes de Datos para su tratamiento posterior".
  - Dicho tratamiento se realiza en bloques ininterrumpidos de proceso, producto de los cuales se obtienen "Lotes de Resultados".

- **PROCESAMIENTO INTERACTIVO (TIEMPO REAL).**
  - Cada unidad lógica de información es introducida al sistema para su tratamiento inmediato y los resultados de dicha acción reportados también en forma inmediata.

LOCALIZACION

- **PROCESO LOCAL.**
  - Los puntos de entrada y salida de resultados se encuentran en el mismo sitio donde se localizan los procesadores que ejecutan el tratamiento de la información.

- **TELEPROCESO.**
  - Los dispositivos de entrada de datos, tratamiento de la información y salida de resultados se encuentran geográficamente dispersos.

GRADO DE CONCENTRACION

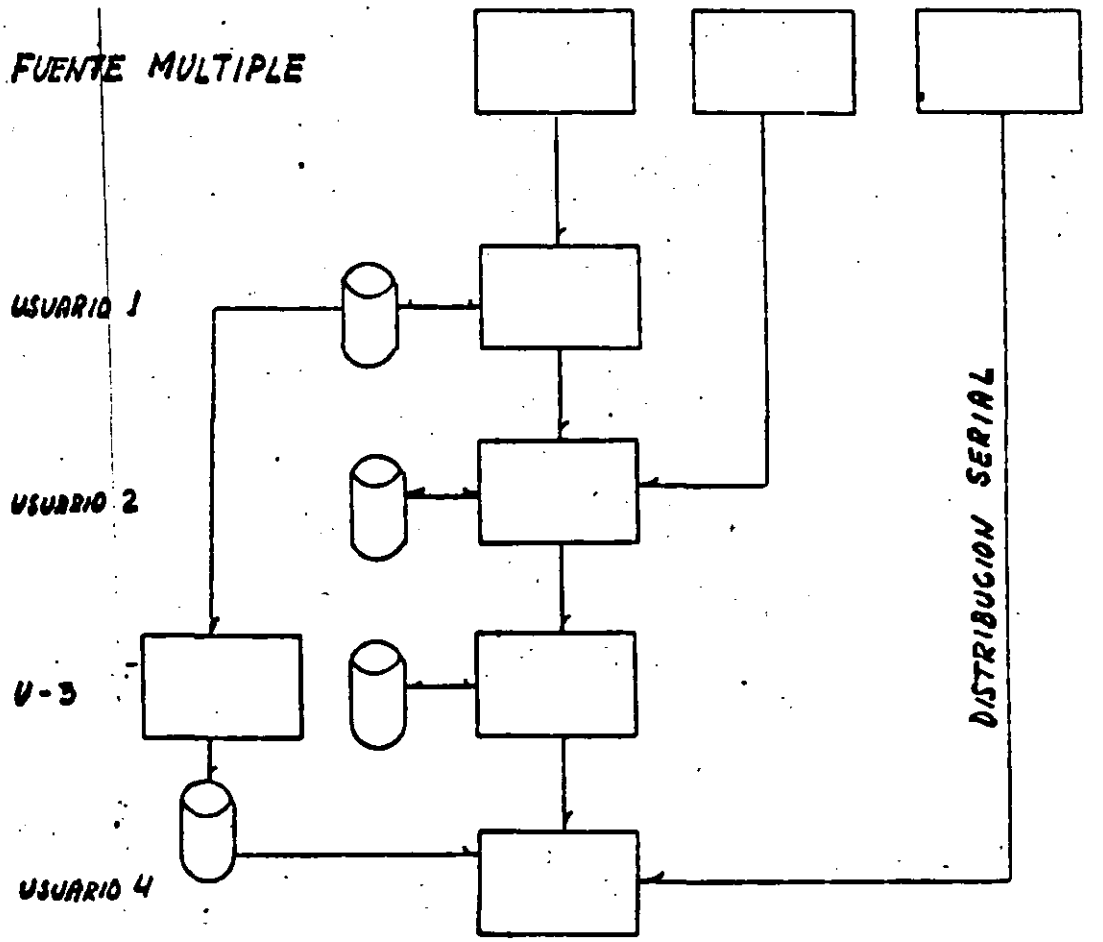
- **PROCESO CENTRALIZADO.**
  - **Los procesos de tratamiento de la información son ejecutados en un sólo punto denominado "Procesador Central" ("Main Frame").**

- **PROCESO DISTRIBUIDO.**
  - **El proceso de tratamiento de la información es realizado por etapas con diferentes unidades de proceso.**

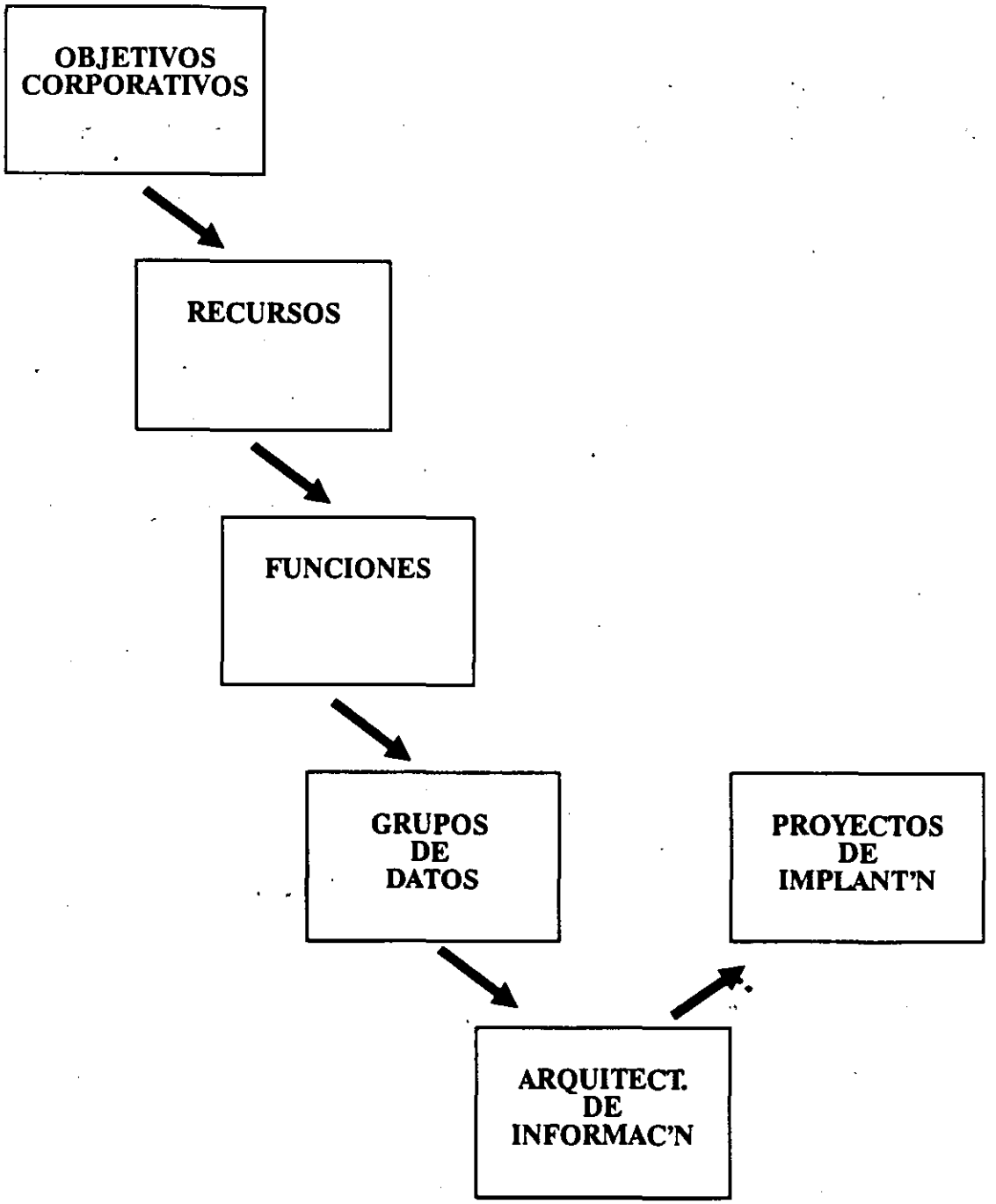
CARACTERISTICAS DEL ENFOQUE USUAL

- Orientados al organigrama (reportes) más que a funciones (datos).
- No cubren necesidades globales solo las de una unidad operacional.
- Propicia fragmentación, repetición, inconsistencia.
- Sistemas rígidos, costosos de desarrollar y mantener.
- Sistemas baratos, prácticos, políticos.
- Enfasis en sistemas operacionales.

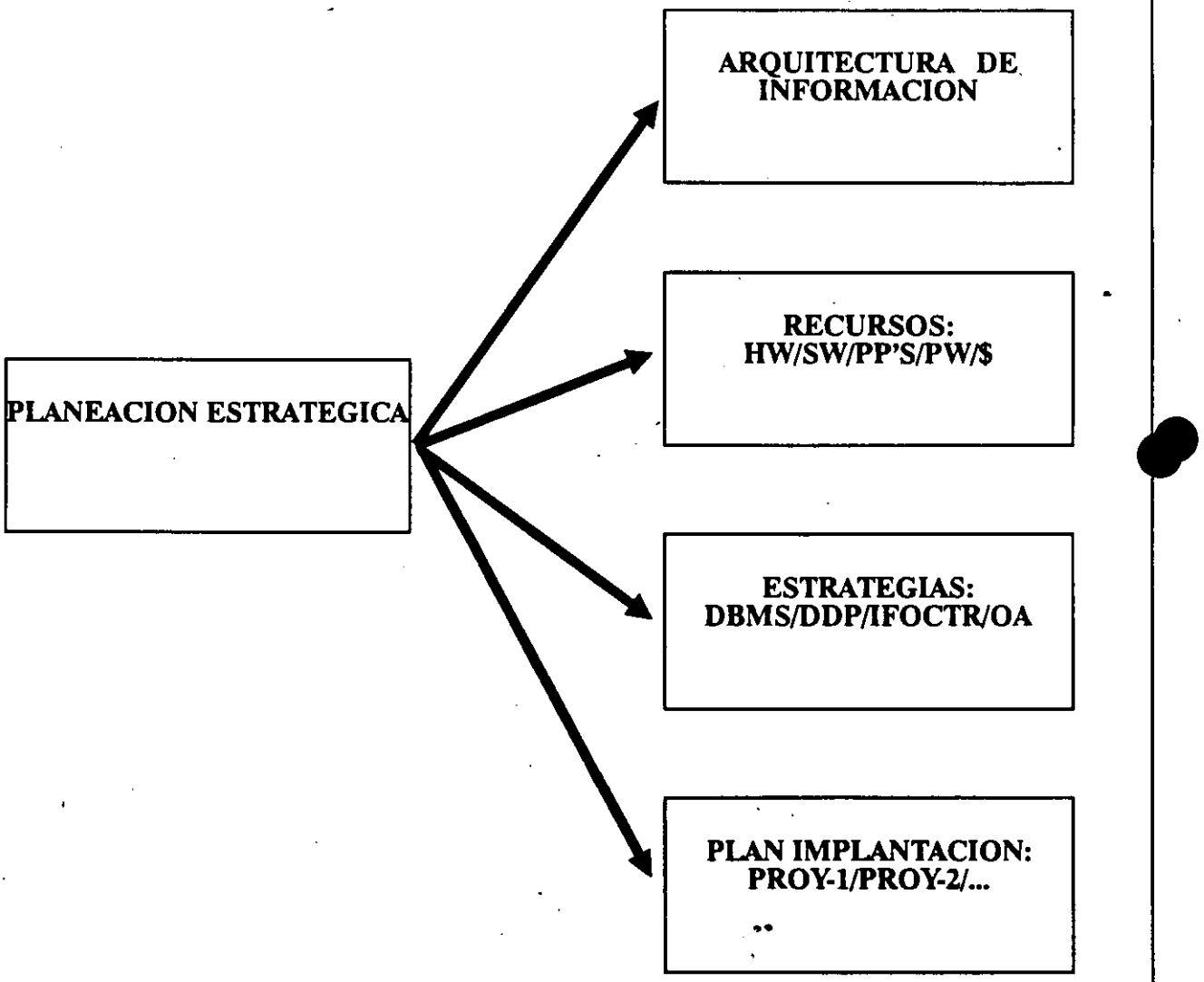
DATOS BAJO DISEÑO TRADICIONAL



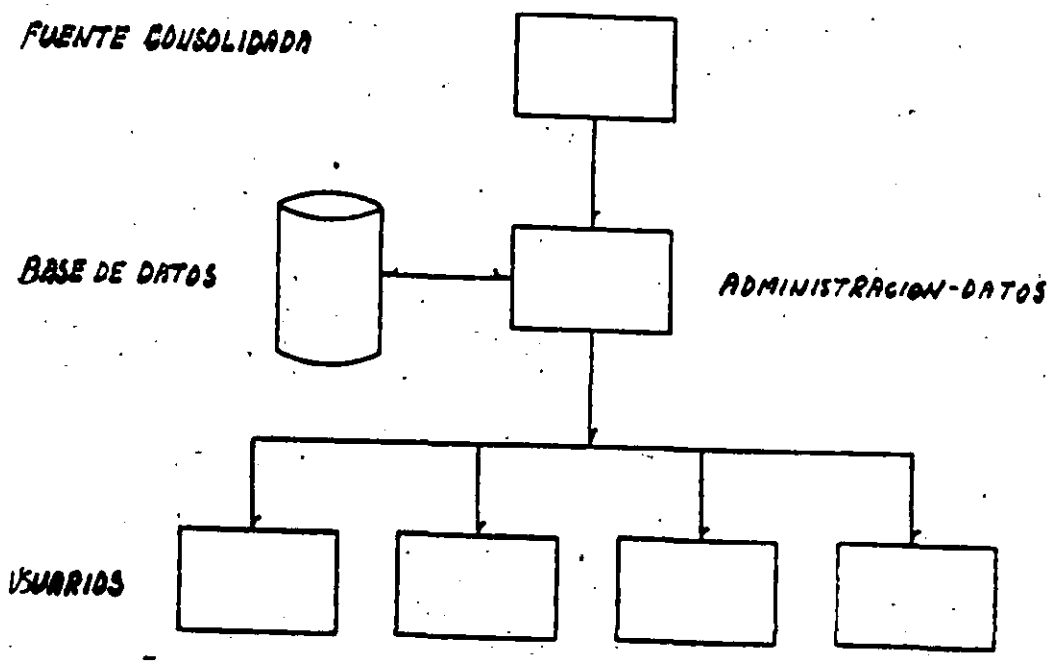
ENFOQUE DESCENDENTE



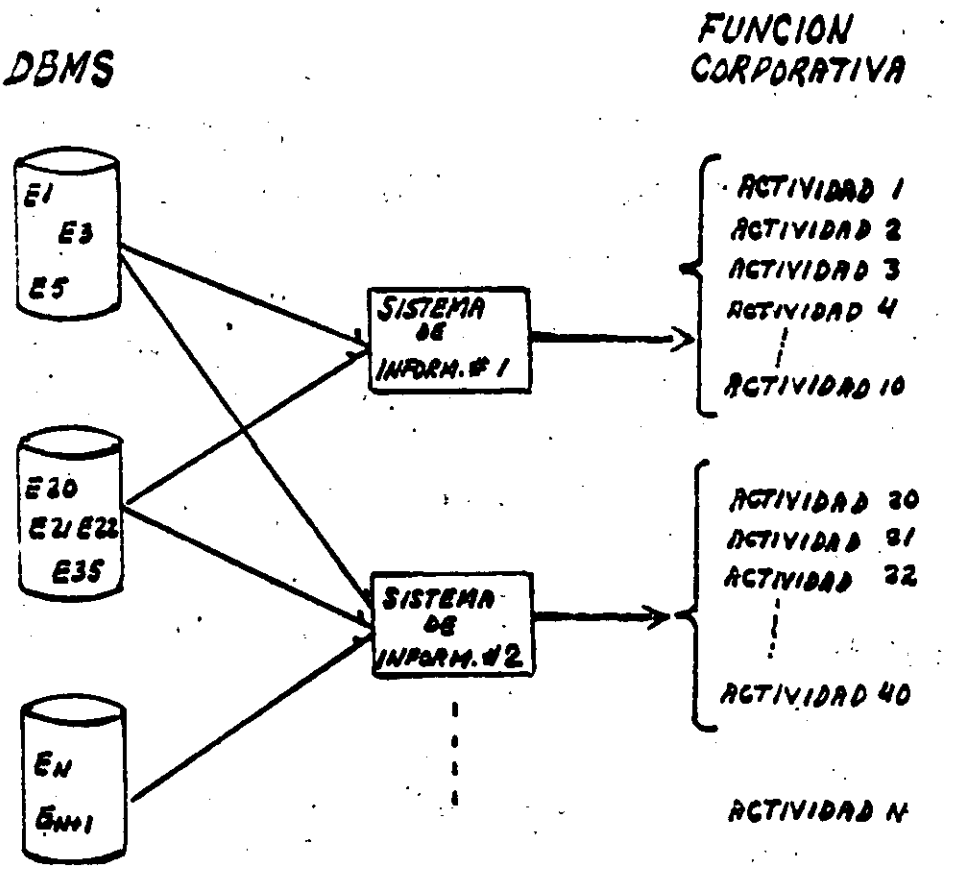
PLAN DE INFORMATICA



DATOS BAJO NUEVO DISEÑO



DISTRIBUCION PARALELA


DBMS
FUNCION CORPORATIVA
E1
E3
E5
E20
E21 E22
E35
EN
EN+1
SISTEMA DE INFORM. # 1
SISTEMA DE INFORM. #2
ACTIVIDAD 1
ACTIVIDAD 2
ACTIVIDAD 3
ACTIVIDAD 4
ACTIVIDAD 10
ACTIVIDAD 20
ACTIVIDAD 21
ACTIVIDAD 22
ACTIVIDAD 40
ACTIVIDAD N

DEFINICIONES.

- **DBMS**
  - **Sistema cuyo objetivo es registrar y mantener datos relevantes a la organización necesarios en los procesos de toma de decisión.**

- **BASE DE DATOS.**
  - **Colección de datos operacionales estructurados que son compartidos por sistemas de aplicación.**

- **DATOS OPERACIONALES (NO E/S).**
  - **Referentes a entes de interés corporativo y sus relaciones (Proceso dado).**

BASE DE DATOS

Conjunto ordenado de archivos de información consolidada, estructurada y normalizada (no redundante), que permite el acceso y actualización de la información en forma simultánea, sincronizada y consistente a uno o varios procesos de usuario.

NOTA: Es importante distinguir los conceptos de "Banco de Datos" y "Base de Datos".

D B M S .- DATA BASE MANAGEMENT SYSTEMAS

(SISTEMAS DE ADMINISTRACION Y CONTROL DE BASES DE DATOS)

Se define como DBMS al conjunto de rutinas, funciones métodos de acceso, áreas de almacenamiento, de trabajo y de control requeridas para el tratamiento de la información bajo el concepto de Base de Datos.

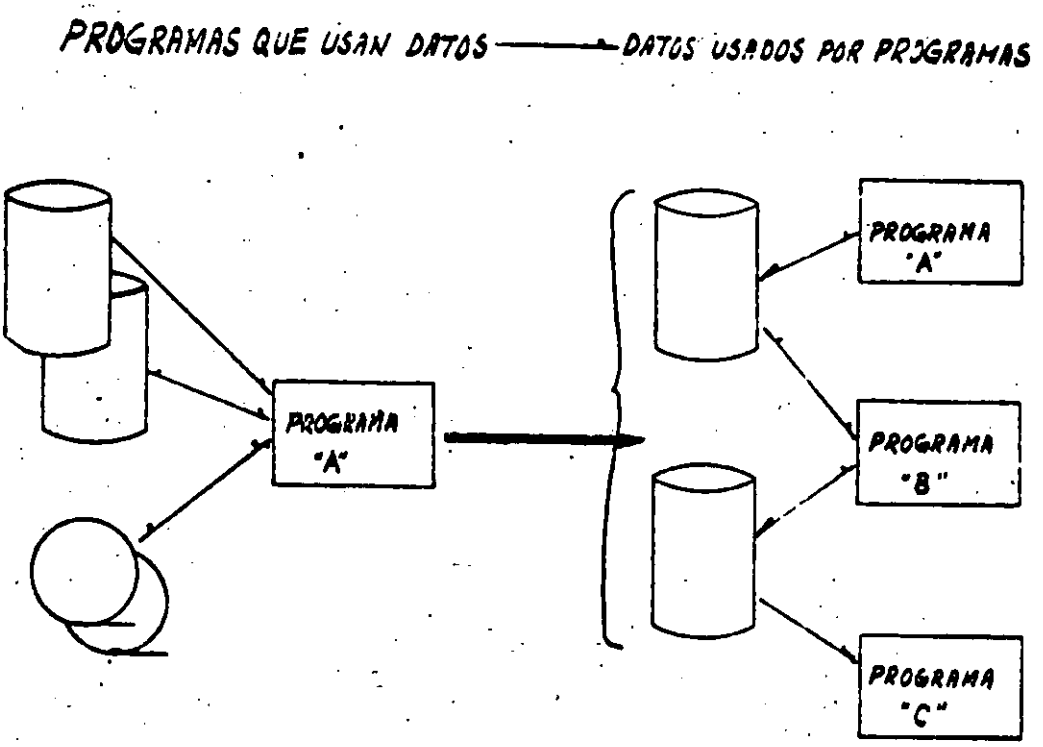
UNA BASE DE DATOS DEBE:

- Contener los datos sin considerar dónde o cómo se almacenan.
- Contener los datos en su forma más simple (no elaborado).
- Ser facilmente expandible.
- Ser accesible.
- No contener reduncancias.
- Ser independiente de las aplicaciones.
- Integridad en los datos, seguridad y mecanismo de recuperación.

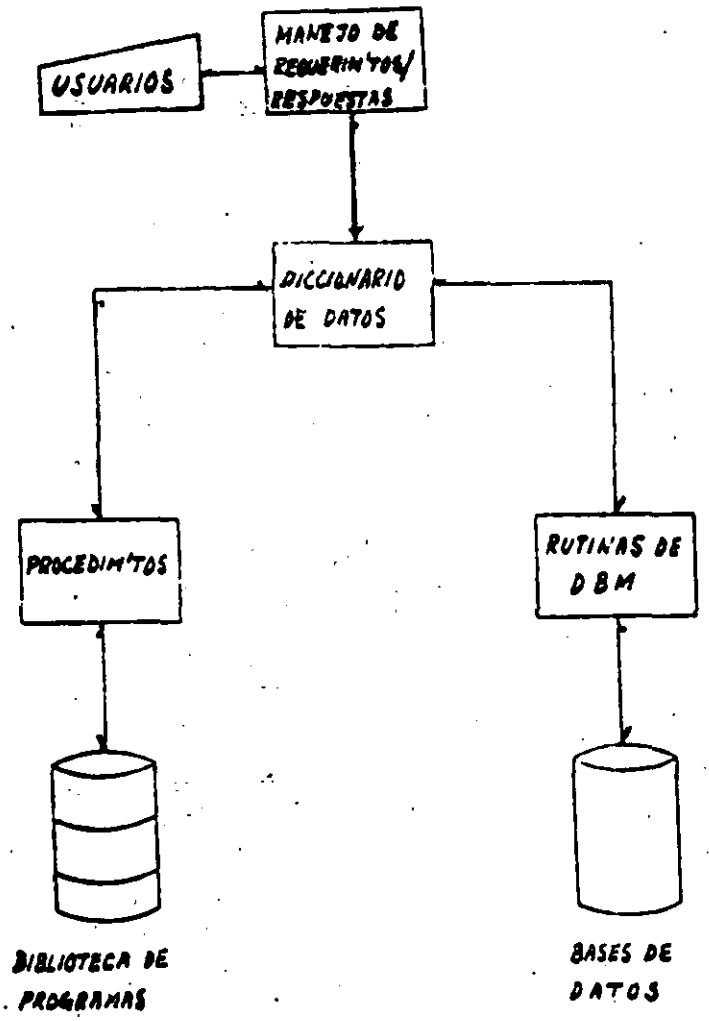
EL PORQUE DE UN D B M S

- **Porqué permite controlar, centralizar y consolidar la información.**
- **Facilita el uso compartido de la información por múltiples usuarios en tiempo real.**
- **Proporciona seguridad en la protección y recuperación de la informaicón.**
- **Aumenta la facilidad de uso de los sistemas de procesamiento de información.**
- **Reduce los requerimientos de procesos de mantenimiento a la información.**
- **Reduce el tiempo requerido para el desarrollo de aplicaciones.**
- **En general, incrementa la eficiencia en el uso de un procesador desde el punto de vista de los usuarios.**

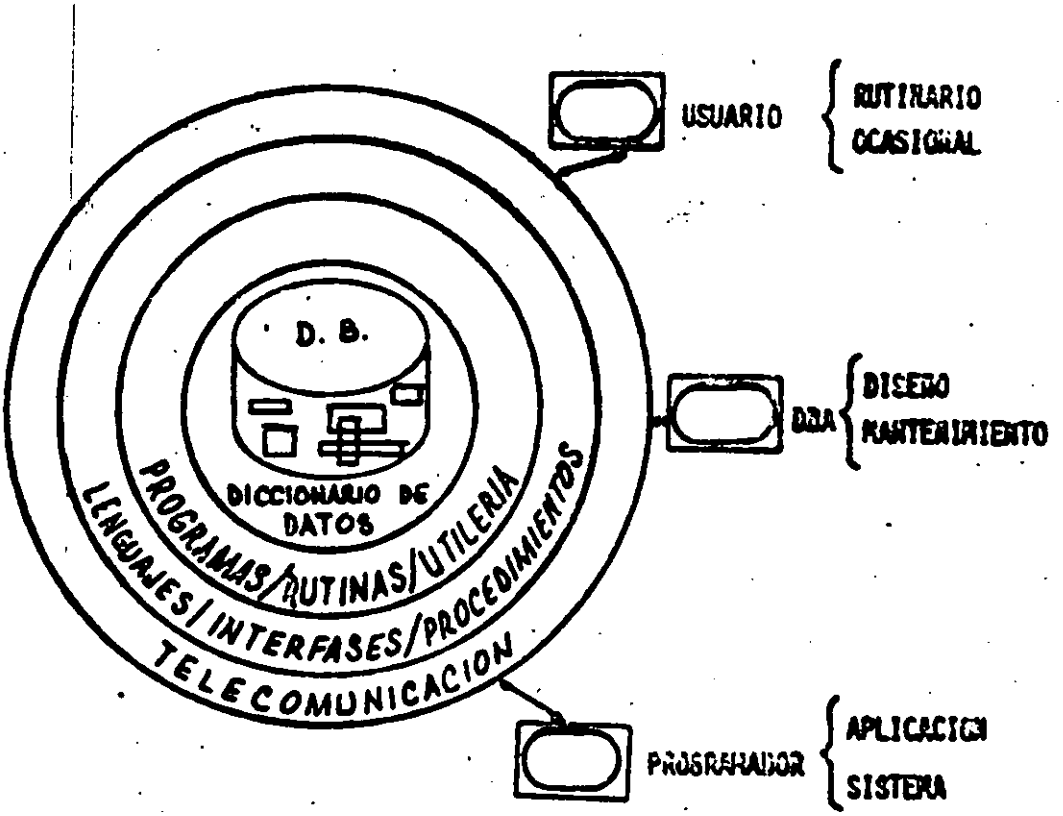
"DBMS" COMO CAMBIO DE ORIENTACION



ESQUEMA DE DBMS



DATA BASE MANAGEMENT SYSTEM (DBMS)

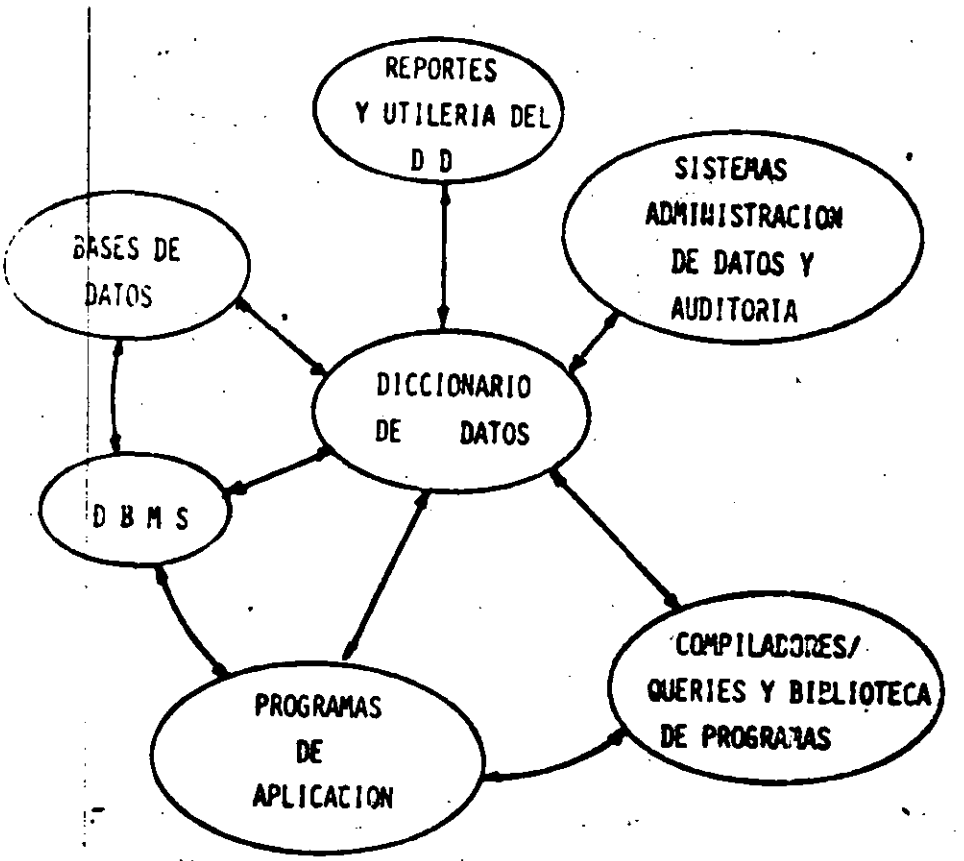


ES EL SISTEMA CUYO CONJUNTO DE RUTINAS, UTILERIA, PROCEDIMIENTOS, INFERFASES, "FIRMWARE" ; MANEJAN CONTROLAN, A PETICIÓN DE LOS VARIOS USUARIOS, LOS "MODELOS DE DATOS" DE LA EMPRESA.

DATA DICTIONARY (D D)

- Subsistema del DBMS. Base de datos que contiene "METADATA".
- Herramienta para controlar los datos mismos.
- Maneja la documentación y otras categorías.
- Debe usarse desde el inicio del primer proyecto.
- D D integrado:
  - Verifica todas las descripciones antes de ejecución.
  - Refreja automaticamente cambios en las tablas.
- Ventajas del DDI:
  - No redundancia del "DDL".
  - Misma fuente obligatoria para todos los usuarios.
- Ventajas del DD-No-I:
  - Menor riesgo de fracaso.
  - Independencia/flexibilidad respecto al DBMS

INTERFASES DE UN D D I



NOTA: Debe soportar todos los modelos: Externo, Conceptual, Interno.

CARACTERISTICAS DE DBMS

- Controla centralmente el recurso corporativo: D A T O S.
- Distribuye los datos en forma paralela.
- Separa "Formato de Datos" de "Uso de Datos".
- Fundamenta la nueva generación de lenguajes.
- Maneja los datos más técnicamente (no artesanalmente).

VENTAJAS DE DBMS

- Control centralizado del recurso datos.
- Distribución paralela de datos, no serial.
- No duplicación = Integridad + Economía.
- Separación del Formato del Uso.
- Facilita el Diseño (no sort-Merge, Back-Up).
- Base para mejores sistemas de información.
- Independencia de datos (lógico vs. Físico).
  - Transferencia a cambios (equipo, sist. op., organización)
  - Facilidad de uso (solo nombres y valores).
  - No necesidad de "navegación" del usuario.

ENTES/MODELOS

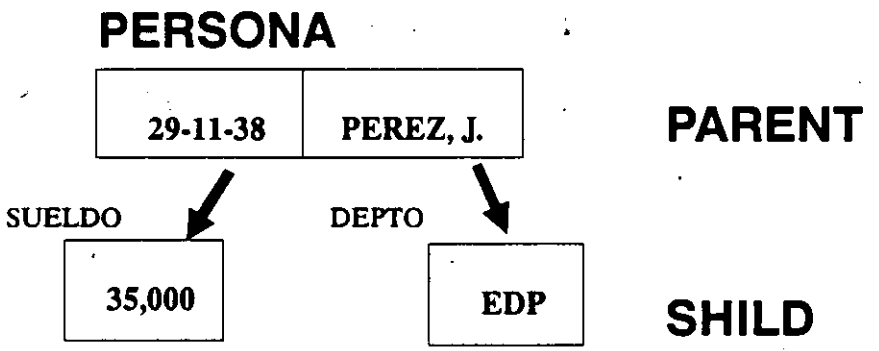
- 'ENTES' son objetos acerca de los cuáles la empresa desea registrar atributos y relaciones.
- Los atributos de los entes que participan en un proceso representan una estructura o modelo de datos.
- Un ente tendrá tantos modelos como procesos en que intervenga.
- Diferentes tipos de DBMS "Modelan" los datos en diversas maneras
  - Tablas
  - Jerarquías
  - Redes

TIPOS DE DBMS

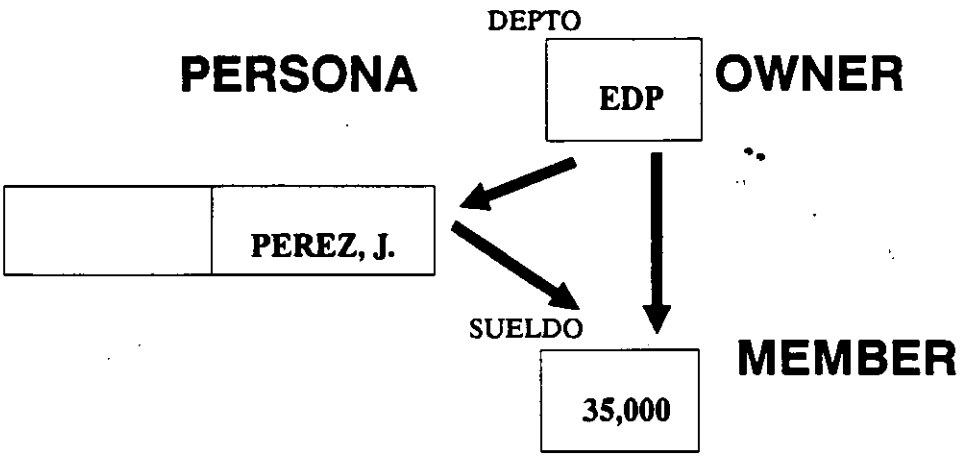
- RELACIONAL

NACIM.	NOMBRE	SUELDO	DEPTO
9-11-3	PEREZ, rJ.	35,000	EDP

- JERARQUICO



- RED
-



CARACTERISTICAS DEL SISTEMA RELACIONAL

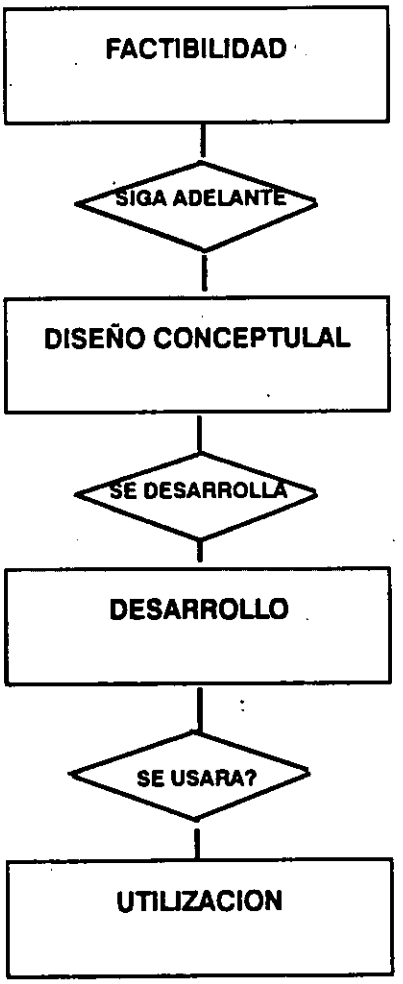
"FUNDAMENTO PRACTICO DE PRODUCTIVIDAD"

E.F. CODD

- Modelo de datos más adecuado a DDP.
- Procesamiento simultáneo de multiples CONJUNTOS DE DATOS.
- Direccionamiento posicional ---Direccionamiento Asociativo.
- Lenguaje común entre profesionales DP y profesionales usuarios.
- Avance hacia una ciencia informática.
- Visitas dinámicas en forma tabular.
- Clave en los lenguajes de 4a.generación

DESARROLLO DE UN SISTEMA (ENFOQUE ANTIGUO)

- Todo está pre-especificado
- Los cambios son muy difíciles (Imposibles)
- La seguridad se asume



DESARROLLO DE UN SISTEMA (4GT)

- El cambio es posible.
- No se necesita pre-especificar.
- Seguridad a través de las herramientas.
- Desarrollo en contacto con el usuario.
- Prototyping.

CONVERSION DE APLICACIONES

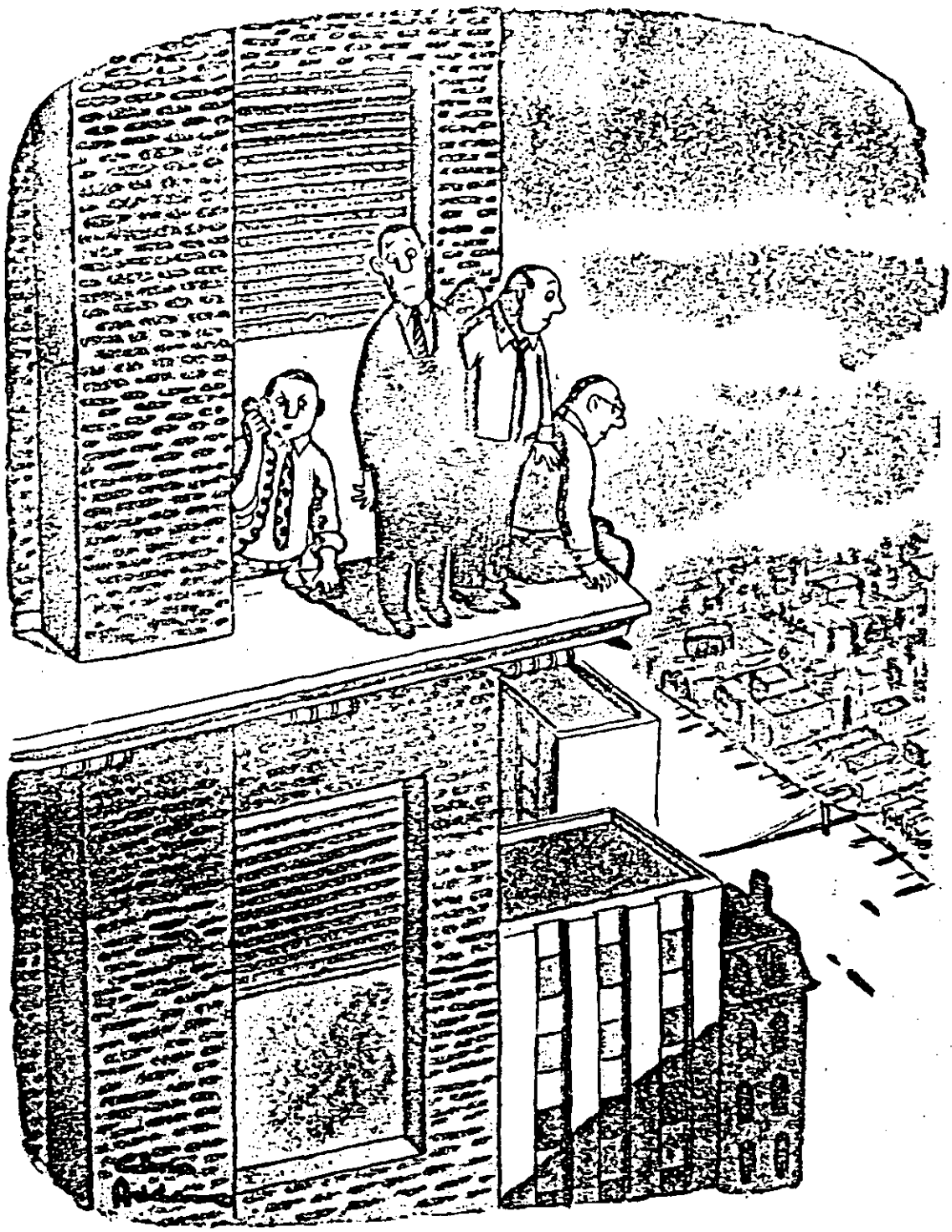
Uno de los más grandes problemas para losusuarios que quieren migrar a un ambiente de 4gt es "que hago con todas mis viejas aplicaciones mientras todo es reescrito en un nuevo codigo?"

TRES TIPOS DE AYUDAS:

1.- Productos que generan copy-codes para ser insertados en los sistemas existentes: copy-code para cobol,pli, etc.

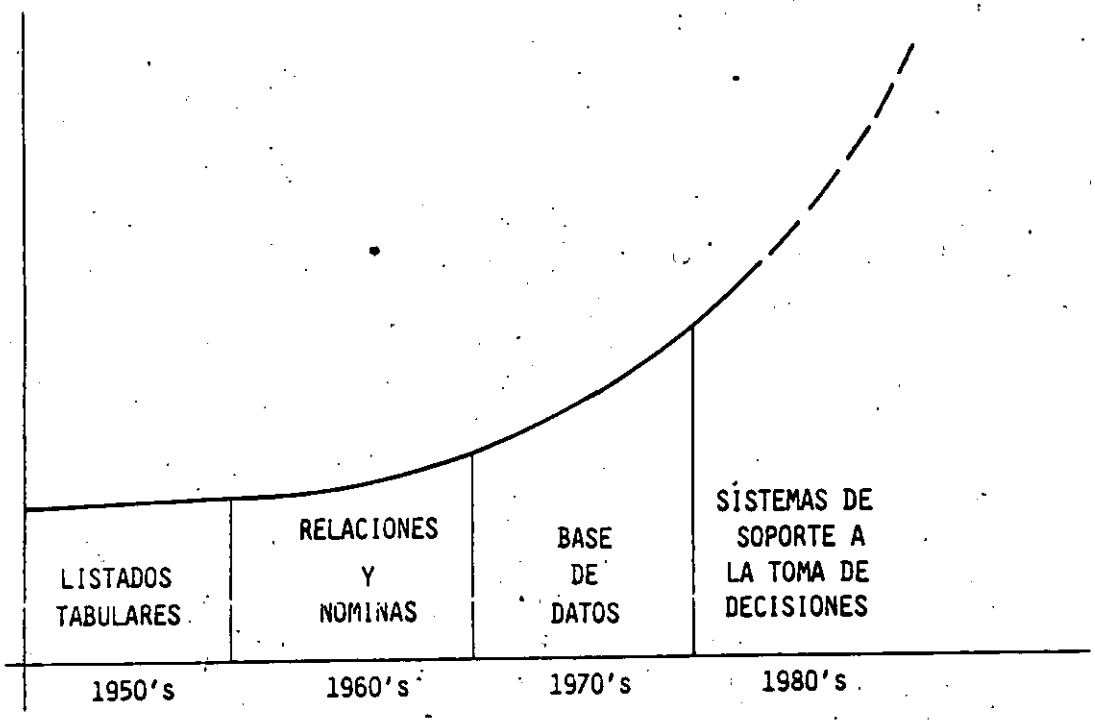
2.- Productos que soportan el modo nativo: los archivos permanecen donde estan y las nuevas herramientas pueden accesar estos archivos.

3.- Productos que soportan el modo transparencia: los archivos son cargados en la nueva base de datos y todas las llamadas de los programas existentes a los archivos son interceptadas y traducidas al manjador de la base de datos.



"About that data base conversion, sir.
The department doesn't seem to share our enthusiasm."

PAPEL DE LA INFORMATICA EN LA ORGANIZACION



SISTEMAS DE SOPORTE DE DECISIONES

Son una combinación de herramientas como base de datos, análisis estadístico, modelos financieros, queries y reporteadores diseñadas para soportar modelos interactivos y situaciones de "qué para si..."

Las aplicaciones tradicionales trabajan con el "qué ha sido" Las aplicaciones de soporte de decisiones trabajan con el "qué será o qué pasara si..."

Las facilidades o lenguajes de simulación son un requisito, así como hojas de cálculo, graficadores, procesadores de texto, correo electrónico y una interfaz micro-mainframe.

- **COMPONENTES:**
  - **Lenguajes no procedurales.**
  - **Lenguajes procedurales.**
  - **Manejador de base de datos.**
  - **Acceso o otras DBMS.**
  - **Generador de reportes.**
  - **Queries.**

CONCLUSION

El objetivo es:

"Proporcionar la información suficiente que, combinada con un criterio adecuado, de como resultado una decisión acertada y oportuna".

PRINCIPIO BASICO DE LOS SISTEMAS DE SOPORTE A LA TOMA DE DECISIONES

"La persona que espera contar siempre con TODA la información necesaria para tomar una desición..."

"NUNCA DECIDIRA NADA"

PROTOTIPOS

- Identificar los requerimientos básicos del usuario por medio de un análisis rápido e incompleto.
- Desarrollar la base de datos (si es relacional, hacerlo sin miedo ya que los cambios son fáciles de hacer).
- Planear y desarrollar menús y submenús.
- Completar el prototipo desarrollando función por función.
- Después de la aprobación del usuario, escribir las especificaciones (documentación).
- Implantar y usar el prototipo.
- Revisar y mejorar el prototipo.

CENTRO DE INFORMACION (INFORMATION CENTER)

La meta de un centro de información es el usar más productivamente las cualidades del departamento de DP.

Esto se hace enseñando a los usuarios finales a desarrollar sus propias aplicaciones y esto se puede hacer cuando se crea un amiente que soporte ésta meta.

El manejo apropidado es más importante para el exito de un centro de información que con que herramienta de Software se desarrolle. Obviamente un centro de información no funcionaria sin 4gl.

El centro de información saca ventaja de la "amigabilidad" de los 4gl para poder ser enseñados a los usuarios finales.

El departamento de DP debe proveer el Hardware, Software,, Bases de Datos y entrenamiento.

Un centro de información exitoso debería estar formado por usuarios de todos los departamentos y un asesor del departamento de DP.

SOFTWARE DBMS Y 4GT

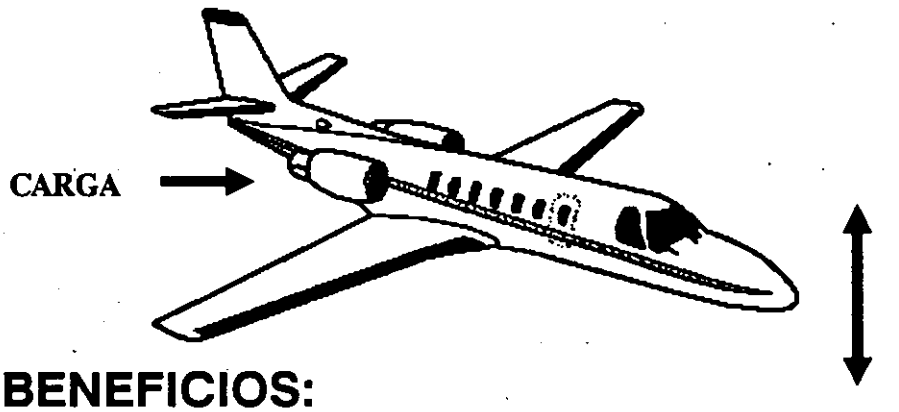
HARDWARE	DBMS	4GT
BURROUGHS	DMSII	LINC II
DATA GENERAL	DG-DMS	DG/SQL
	ORACLE	SQL
		POWEROUSE
VAX	RDB	RALLY
	ULTRA	MANTIS
	ORACLE	SQL
	INGRES	QUERY
		POWERHOUSE
	ADABAS	NATURAL
HEWLETT PACKARD	IMAGE 3000	QUERY 3000
		POWERHOUSE
		SPEEDWARE
WANG	PACE	QUERY
	THE ANSWER	QL/1
PRIME, GOULD	CENTRE	
IBM	ORACLE	SQL
	SUPRA	MANTIS
	INGRES	QUERY
	DATACOM	IDEAL
	MODEL 204	WORKSHOP
	IDMS	ADS
	IMS	DL/1
	ADABAS	NATURAL

CASO SWISSAIR

PROBLEMA:

- En el año de 1980 la Compañía Swissair (Línea Aerea Suiza) detectó que debido a los niveles salariales existentes en Suiza y sus tendencias de evolución (superiores al resto de Europa), sus costos de operación resultarían sistemáticamente superiores a las otras Líneas Aereas.
- El reflejar esto en las tarifas los pondría fuera de competencia frente a las otras Líneas Aereas.
- El reducir el beneficio a los acccionistas los pondrían en desventaja frente a otras posibilidades de inversión y podría traer como consecuencia serias fugas de inversionistas.
- El repercutirlo en el nivel de calidad del servicio los pondría en seria desventaja frente a las otras Líneas Aereas.

SOLUCION
SITEMA AUTOMATIZADO "LOAD & BALANCE"



BENEFICIOS:

- **Reducción del tiempo de utilización de platafor ma de abordaje. (Alquiladas por los Aeropuertos).**
- **Reducción de la tolerancia en los horarios de pis ta de despegue. (Alquiladas por los Aeropuertos).**
- **Reducción de las operaciones de carga y descarga.**
- **Máximo aprovechamiento de los compor-tamientos de carga.**
- **Reducción de la pérdida de tiempo provocada al pasajero.**

PLANTEAMIENTO

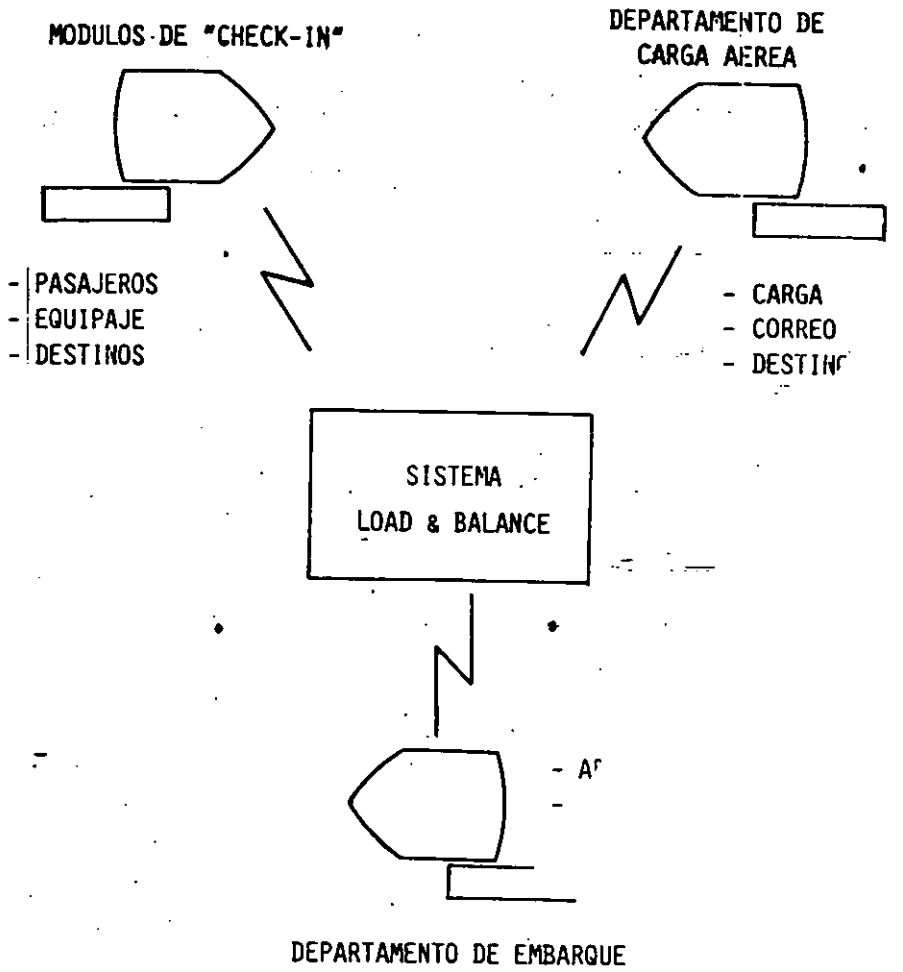
Objetivo:

- **Reducir en al menos $2'000,000.00 US. Dlls. al año los costos de operación en tierra de aeronaves, a partir del año de 1982.**

Condicionantes:

- **Sin reducir su nivel de calidad en los servicios.**
- **Sin reducir su volumén de operaciones.**

DESCRIPCION GENERAL



CONTROL DE PROYECTO

- El proyecto siempre fué de la Vicepresidencia de Operaciones en tierra.
- La herramienta la proporcionó Informática.
- El objetivo siempre fué alcanzar el objetivo de negocio: $2'000,000.00 Dlls. al año. + Inversión en el Sistema
- El usuario/patrocinador siempre tuvo el control de "que" y "cuando" poner en operación.
- El objetivo institucional siempre fué "salvar la Empresa".

RESULTADOS

- **Se logró el objetivo Institucional.**
- **Se superó el objetivo de negocio desde el primer año y posteriormente.**
- **Se mejoraron los servicios de pasajeros, equipaje y carga aerea.**
- **Se incrementaron los clientes, volúmenes de operación e ingresos.**
- **En 1982 se obtuvo un premio a la excelencia en el servicio de la Asociación de Líneas Aereas.**
- **Se incrementó el prestigio de la Organización.**

IMPORTANCIA

"Lograr convertir un problema de restricción presupuestal en un éxito empresarial y un premio internacional de excelencia en los servicios".

CONCLUSION

Un buen proyecto de sistematización es aquel que ha sido diseñado para que su avance vaya convirtiendo cada "punto crítico" en "factor de exito".

Donde encaja la planeación de un Sistema Integral de información en una organización.

Planeación de Sistemas NO es una función de "Procesamiento de Datos": es una parte del proceso de planeación corporativa de la organización.

OBJETIVO DE UN SISTEMA INTEGRAL DE INFORMACION

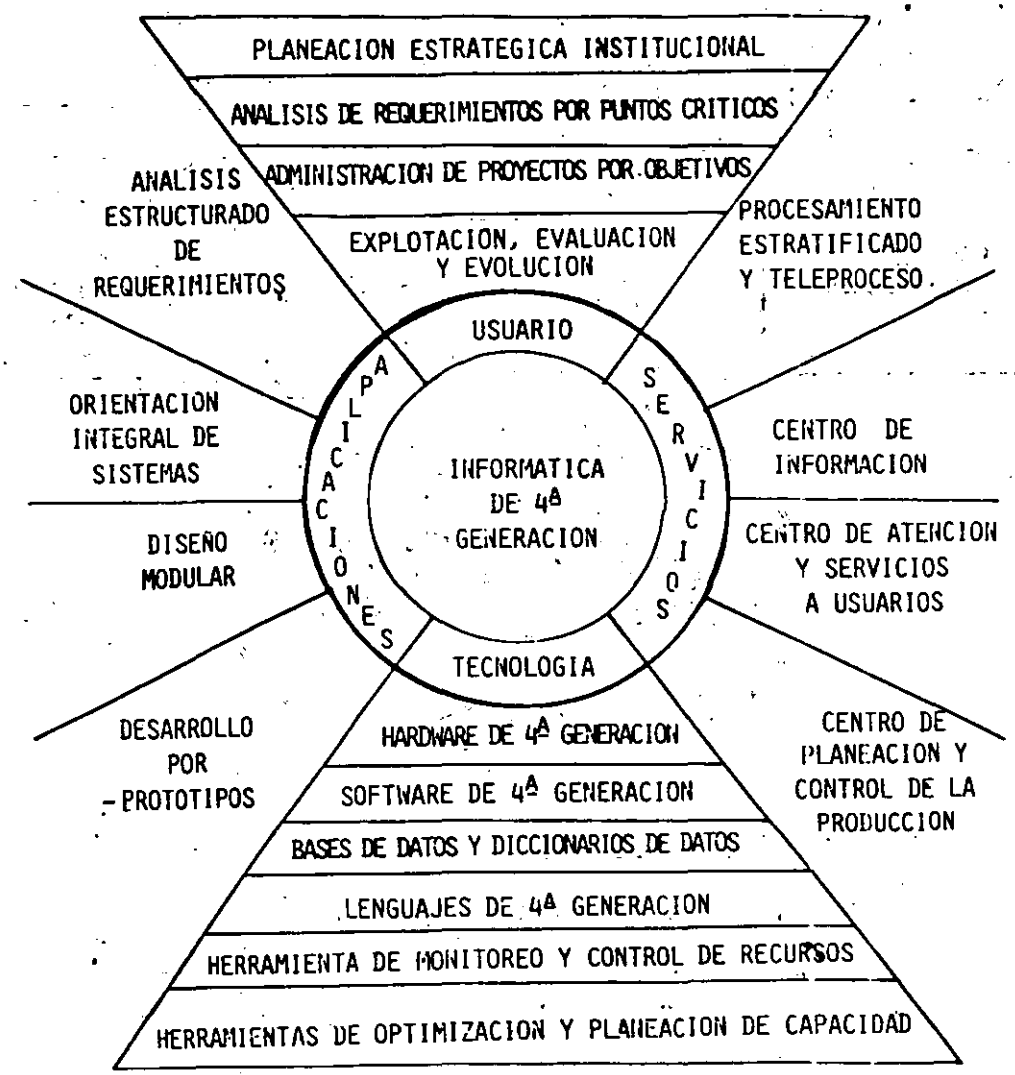
EL OBJETO NO ES:

- **Lograr una convivencia sin conflictos con el único fín de tenerlos juntos.**

EL OBJETIVO ES:

- **Lograr una verdadera integración, a fín de poder aprovechar las características de cada elemento, no solo para su función específica, sino para apoyar y acelerar la evolución total de la organización.**

PRINCIPALES COMPONENTES DE LA INFORMATICA ACTUAL (PUNTOS CRITICOS)



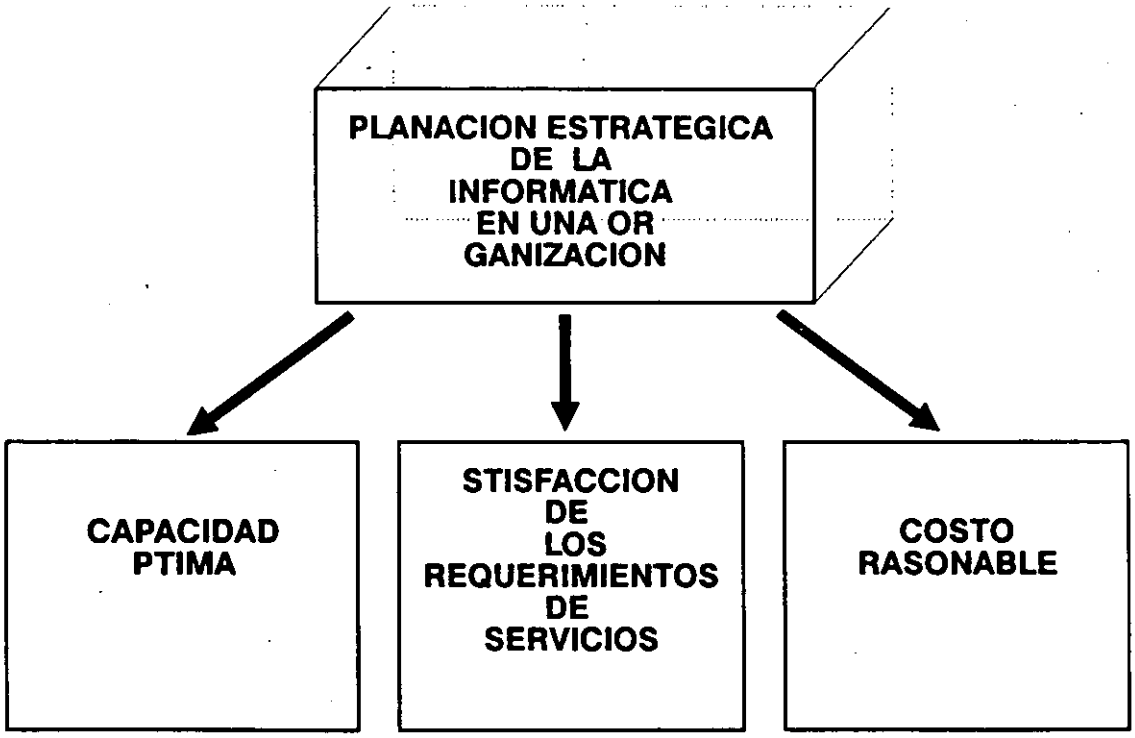
OBJETIVO DE LA PLANEACION ESTRATEGICA EN INFORMATICA

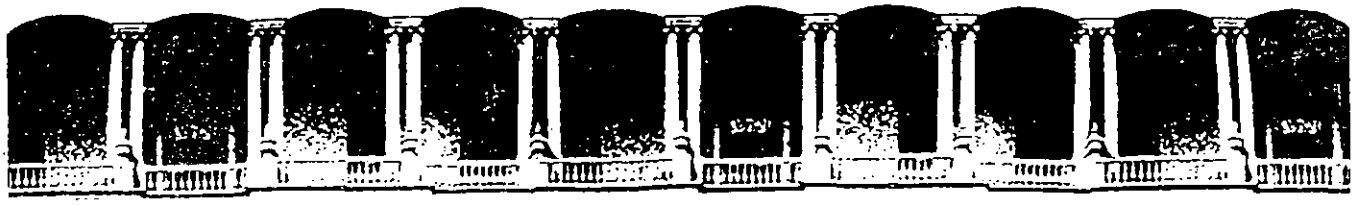
- LOS OBJETIVOS NO SON:
  - Proceso Distribuído Vs. Proceso Centralizado
  - Minis- MicrosVs.Maiframes
  - Sistemas DedicadosVs.Sistemas Compartidos

Esto es evaluación Tecnológica.

- EL OBEJTIVO ES:
  - Proporcionar a la Organización los servicios requeridos para el logro de sus objetivos de manera Segura, Consistente, Sistemática y a Costos Razonables.

RESULTADOS DE UN BUEN PROCESO DE PLANEACION EN INFORMATICA





FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS ABIERTOS

DISEÑO DE BASE DE DATOS

INTRODUCCION A LOS SISTEMAS MANEJADORES DE DATOS

PARTE 2

M. EN C. EFRAIN PARDO ORTIZ

JUNIO
1992

CURSO ABIERTOS
DISEÑO DE BASES DE DATOS
PARTE II

INTRODUCCION A LOS SISTEMAS MANEJADORES DE DATOS

M. EN C. EFRAIN PARDO ORTIZ

mayo, junio, 1991.

SISTEMAS DE ADMINISTRACIÓN DE BASES DE DATOS

La base de datos puede definirse, como una colección de datos interrelacionados almacenados en conjunto sin redundancias perjudiciales o innecesarias ; su finalidad es la de servir a una aplicación o más, de la mejor manera posible; los datos se almacenan de modo que resulten independientes de los programas que los usan; se emplean métodos bien determinados para incluir datos nuevos y para modificar o extraer los datos almacenados.

Evolución del Concepto de Base de Datos.

La expresión base de datos comenzó a popularizarse al principio del decenio que se inicio en 1960.

Antes de esa época, en el mundo de la informática se hablaba de archivos y conjuntos de datos. Como ocurre a menudo cuando un nuevo término se pone de moda, no faltaron quienes quisieron promover de categoría sus archivos llamándolos bases de datos sin preocuparse por cambiar su naturaleza, como hubiera sido necesario para donarlo de las características de no redundancia, independencia de datos interconectividad, protección de seguridad y en muchos casos, accesibilidad en tiempo real.

Antes de que aparecieran las computadoras de la tercera generación, la mayoría de los archivos se organizaban de manera secuencial simple.

El Software ejecutaba las operaciones de entrada/salida de los dispositivos de almacenamiento y un poco más.

La codificación incluida en los programas de aplicación se encargaba de la organización de los datos.

No había independencias de datos. Si se modificaba la organización de los datos o se cambiaban los dispositivos de almacenamiento, el programador estaba obligado a escribir los programas y repetir los procesos de compilación y depuración.

La mayoría de los archivos servían solo para una aplicación. Muchas veces los mismos datos podían servir para otras aplicaciones, pero casi siempre organizados de otro modo lo cual obligaba la creación de otros archivos.

Esto originaba un elevado nivel de redundancia, con varios archivos que en resumen contenían prácticamente los mismos datos.

Posteriormente (Predominantemente al final del decenio; 1960-1970). Se reconoció la naturaleza cambiante de los archivos y de los dispositivos de almacenamiento. En ésta época se intento proteger al programador contra los efectos de los cambios que se producían en el Hardware.

El Software hizo posible modificar la distribución físico de los datos sin que por ello se alterase su estructura lógica.

Los archivos utilizados en esta época por lo general diseñados, como los de la época anterior, para una aplicación determinada o para un grupo de aplicaciones muy similares.

Sin embargo aún persistía la creación de archivos que requiriesen los datos organizados de otra manera, lo cual consecuentemente daba cabida a la redundancia de datos.

Ciclo de Vida de un Sistema de Base de Datos.

Las principales fases del ciclo de vida de un sistema de base de datos son las siguientes:

1.- Diseño de la Base de Datos.

2.- Creación Física de la Base de Datos.

3.- Conversión de los conjuntos de Datos y Aplicaciones Existentes para su acoplamiento a la nueva Base de Datos Creada.

4.- Integración de las aplicaciones convertidas en la nueva base de datos.

5.- Fase de Operaciones.

6.- Fase de Crecimiento, Cambio y Mantenimiento.

Antes de que cualquier base de datos sea diseñada es necesario para una empresa levantar un inventario de todas las áreas en las cuales existen datos, ya sea que estén o no incluidas esas áreas en el sistema de base de datos global.

Este es uno de los pasos importantes en el diseño del sistema de base de datos.

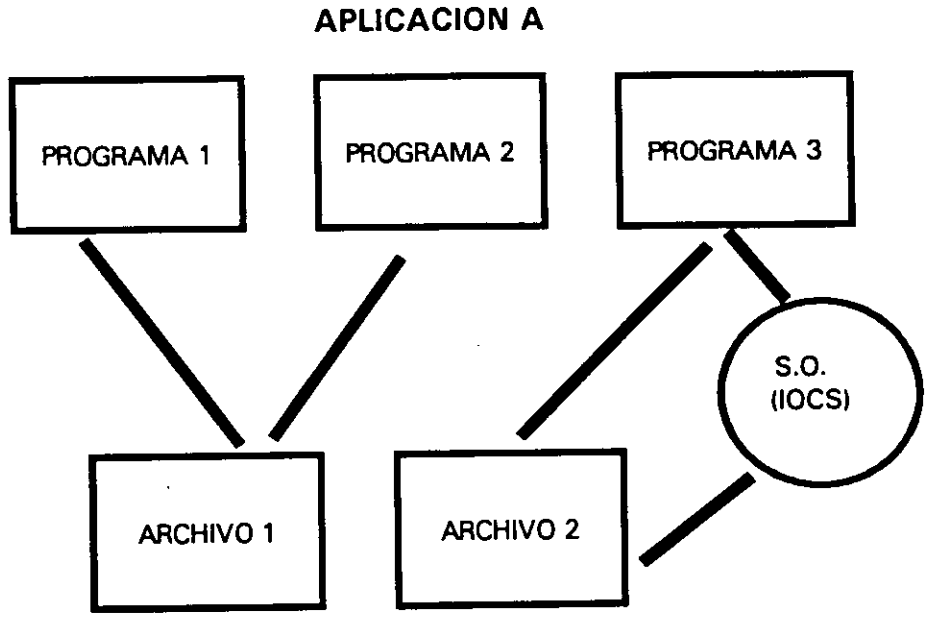
En éste momento se deben revisar los recursos disponibles y comprender los datos, sus recursos, dependencias y relaciones con otros sistemas, así como también se debe desarrollar un esquema de los recursos de datos totales de la empresa y la manera en que ésta información fluye entre los diversos sistemas, para crear un plan de diseño del sistema.

El plan de diseño del sistema servirá como una guía para asegurar que el sistema se esta desarrollando de manera ordenada.

Este plan establecerá la dirección que la empresa tomará, por lo que debe ser completo y dirigido a todos los sistemas.

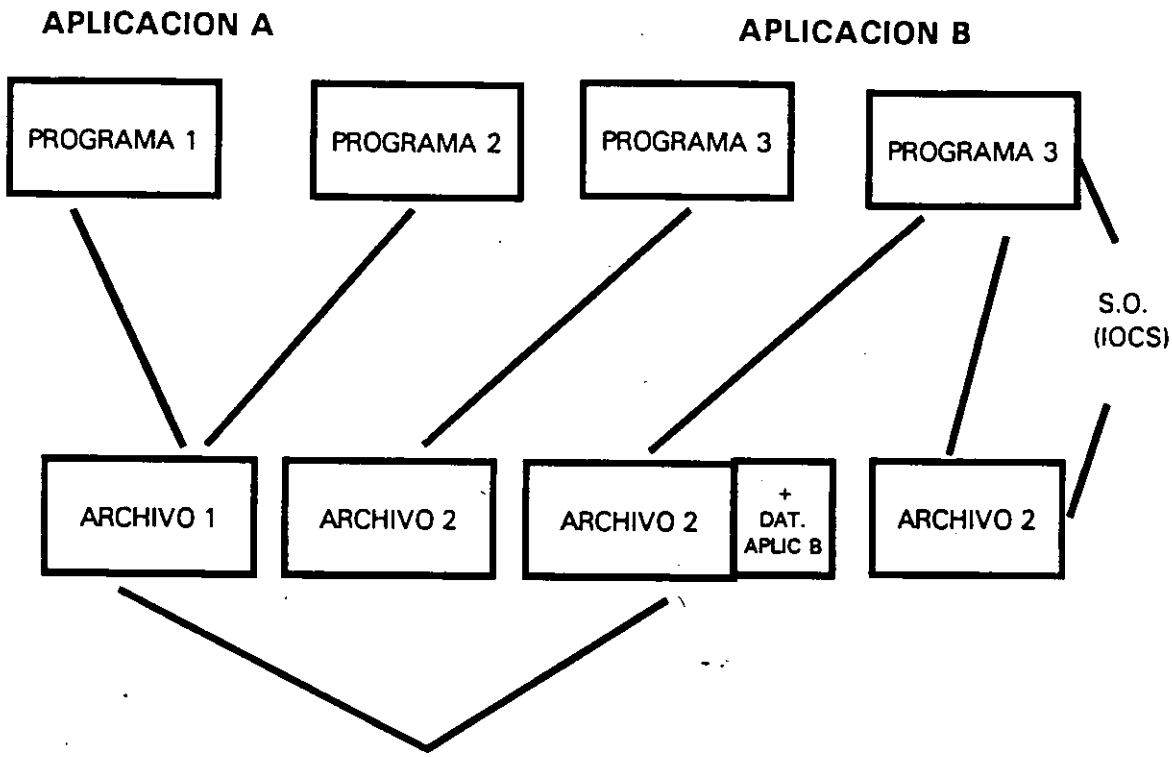
EVOLUCIÓN DEL CONCEPTO DE BASE DE DATOS

Arquitectura de un sistema manejado or IOCS



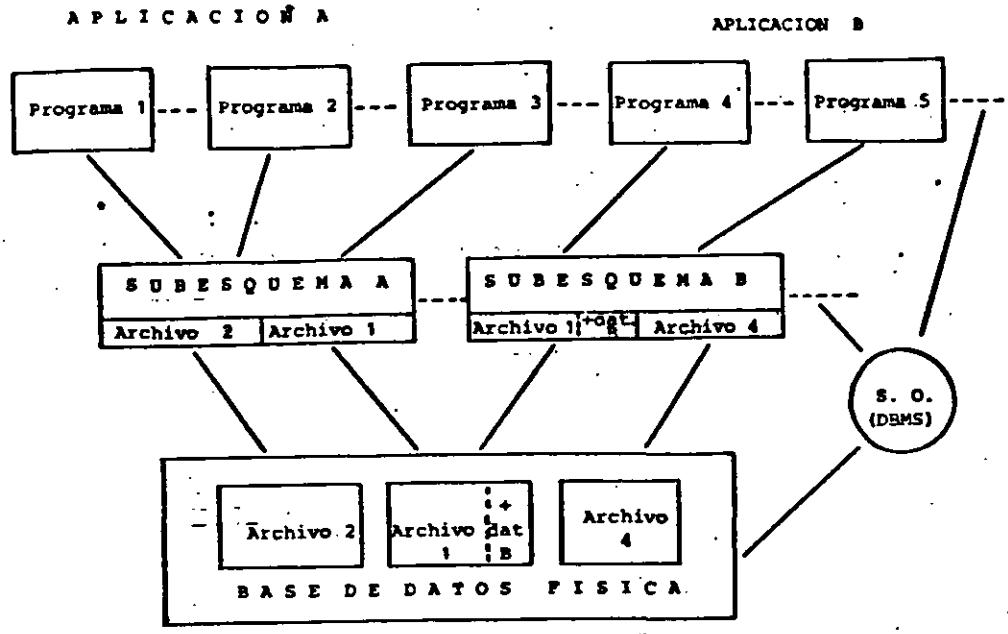
Funciones de Programa	Funciones de Sistema Operativo
• **Instrucciones sencillas para las operaciones deEntrada/Salida.** • **Métodos de acceso a los archivos.** • **Rutinas para Selección de Registros por archivo.** • **Funciones Específicas de cada programa;**	• **Operaciones de Entrada/Salida.**

Arquitectura de un Sistema manejado por FMS



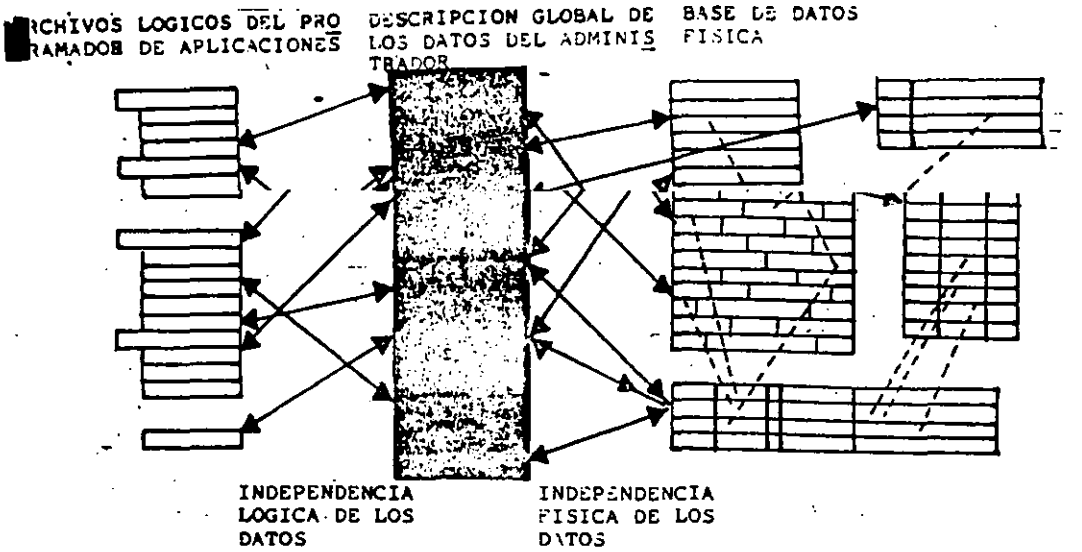
Funciones de Programa	Funciones de Sistema Operativo (File Management System)
• **Instrucciones sencillas para las operaciones Entrada/Salida.** • **Instrucciones sencillas para el manejo de los métodos de acceso.** • **Rutinas para la Selección de Registros** • **Funciones específicas para cada programa..**	• **Rutinas para Métodos de Acceso.** • **Operaciones de Entrada/Salida.**

Arquitectuara de una Basse de Datos con Subesquemas



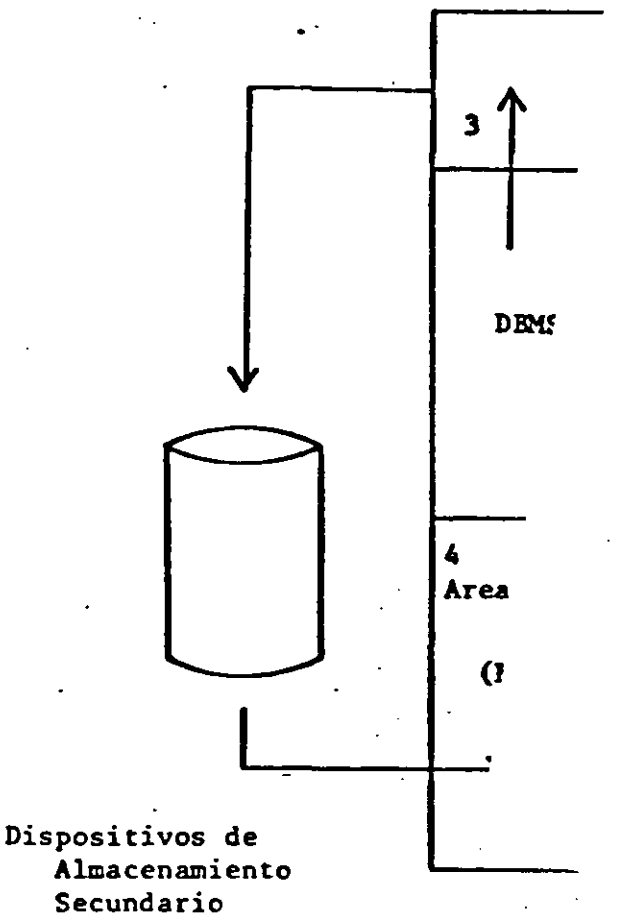
Funciones de Programa	Funciones del Sistema Operativo
• Instrucciones sencillas para Selección de registros • Instrucciones sencillas para Acceso a los diferentes archivos. • Funciones Específicas de cada Programa.	• Rutinas de Selección de Registros. • Rutinas para Métodos de Acceso. • Operaciones de Entrada/Salida.

ETAPA 4: PRIMERAS BASES DE DATOS (PREDOMINATES AL PRINCIPIO DEL DECENCIO 1970-1980



CARACTERISTICAS

- INDEPENDENCIA FÍSICA Y LÓGICA DE LOS DATOS.
- VISIONES GLOBALES INDEPENDIENTES DEL CONJUNTO DE DATOS.
- CONTROL Y CUSTODIA DE DATOS A TRABES DE UN ADMINISTRADOR DE DATOS.
- PRIBACIA, SEGURIDAD E INTEGRIDAD DE DATOS.
- LENGUAJE DE DESCRIPCIÓN DE DATOS PARA EL ADMINISTRADOR DE DATOS.
- LENGUAJE DE ORDENES PARA EL PROGRAMADOR DE APLICACIONES.
- LENGUAJE DE INTERROGACIÓN PARA EL USUARIO



Secuencia de eventos cuando un programa necesita un registro lógico.

DISEÑO DE BASES DE DATOS

- Construcción de un manejador de bases de datos.
- Construcción de un sistema de información.

- **Metas.**
  - Modelar y estructurar los datos en forma representativa del mundo real.
  - Implementable en forma eficiente en un sistema hardware/software existente.
- **Modelo de datos.**
  - Representación de los datos y sus relaciones.
- **Niveles de Modelos.**
  - Modelo conceptual
    - Entidades, atributos, asociaciones entre estos.
    - Punto de vista de los directivos de la organización.
  - Modelo Lógico(esquema externo o modelo implementable).
    - Archivos, registros, campos, interpelaciones entre registros y archivos.
    - Punto de vista del programador, usuario final y en en forma global del Administrador de la Base de Datos.
  - Modelo Físico.
    - Archivos, registros, bloques, compresión, apuntadores y directorios.

CICLO DE VIDA DE LAS BASES DE DATOS.

- **Entrevis**
  - Que información se incluye.
- **Diseño.**
  - Modelos conceptual, lógico y físico.
- **Implementación.**
  - Alimentar los datos.
- **Mantenimiento y Afinación.**

¿QUE ES UNA BASE DE DATOS?

- **ES UNA COLECCIÓN DE DATOS INTERRELACIONADOS ALMACENADOS EN CONJUNTO SIN REDUNDANCIAS PERJUDICIALES O INNECESARIAS.**

EXTENCION DE FUNCIONES DE UN DBMS

- **1.- Utilerías.**
  - cargado/descargador.
  - ayudas de diseño.
  - monitoreo de desempeño.
- **2.- Diccionario de datos.**
- **3.- Ayudas para desarrollo de aplicaciones.**
  - procesador de entradas.
  - reportador.
  - lenguaje interactivo (query).

VENTAJAS Y DESVENTAJAS DE UN SISTEMA CON BASE DE DATOS

Existe gran posibilidad de que no todas las organizaciones puedan arriesgarse o de hecho necesiten implementar una Base de Datos. Muchas de ellas probablemente operen más efectivamente con un sistema de archivos secuenciales no relacionados entre sí y un conjunto de programas que los utilicen. La decisión de implantar o no un Sistema con Base de Datos depende principalmente de los requerimientos y objetivos de los usuarios.

Para tomar esa decisión es necesario tener en cuenta las ventajas y desventajas que lleva consigo el implantarla, a continuación se describen algunas de ellas.

Ventajas:

1.- Da la capacidad de organizar los datos de manera conveniente y de acuerdo a las funciones interrelacionada de una organización.

2.- Las descripciones de los datos forman parte de una librería de la Base de Datos, de ésta forma liará a los programas de la administración de los datos haciéndolos más simples.

3.- Permite una integración mayor de los datos para minimizar la redundancia.

4.- Facilita el crecimiento de la Base de Datos sin grandes modificaciones al Sistema.

5.- Da rápidas respuestas a las necesidades de los usuarios.

6.- Permite al usuario con pocos conocimientos sobre computación hacer preguntas a la Base de Datos, sin elaborar complicados programas.

7.- Facilita la introducción de cambios que los usuarios hicieran fuera de tiempo, ya que ellos no pueden anticipar los requerimientos futuros ni asegurar que los que han sido identificados permanezcan constantes.

8.- Las actualizaciones a los archivos ocurren en forma simultánea

9.- Se reducen los errores en los datos y la inconsistencia al minimizar la redundancia.

10.- Puede proporcionarse servicios amplios, más coordinandose información más relevante a los usuarios.

11.- Se efectúa un ahorro en los costos.

12.- La interrogación directa a la Base de Datos permite compactar la salida de reportes voluminosos.

13.- Se reduce el manejo manual de datos ya que se elimina transferencias y datos reentrantes.

Es importante hacer énfasis que en el presente, todas éstas ventajas pueden lograrse solamente con un alto nivel tecnológico. Sin embargo, la implantación de un nivel tecnológico medio.

ORGANIZACIÓN DE LAS BASES DE DATOS

OBJETIVOS SECUNDARIOS (Para facilitar el logro de los objetivos primarios).

- **Independencia física de los datos.**
  - El hardware de almacenamiento y las técnicas físicas de almacenamiento podrán ser modificados sin obligar a la modificación de los programas de aplicación (Fig. 4.1).
- **Independencia lógica de los datos.**
  - Podrán agregarse nuevos ítems de datos, o expandirse la estructura lógica general, sin que sea necesario reescribir los programas de aplicación existentes (Fig. 4.2)
- **Redundancia controlada.**
  - Los ítems de datos serán almacenados una sola vez, excepto cuando existan razones técnicas o económicas que aconsejen el almacenamiento redundante.
- **Adecuada rapidez de acceso.**
  - Los mecanismos de acceso y los métodos de direccionamiento serán lo suficientemente rápidos, rápida cuenta de los usos previstos.
- **Adecuada rapidez de exploración.**
  - La conveniencia y necesidad de la exploración espontánea se incrementarán en la medida que se difunda el uso interactivo de los sistemas.
- **Normalización de los datos dentro de un organismo.**
  - Se necesita un acuerdo interdepartamental sobre los formatos y las definiciones de datos. La normalización entre departamentos es indispensable porque de otro modo ellos crearían datos incompatibles.
- **Diccionario de datos.**
  - Se necesita un diccionario de datos que defina todos los ítems de datos.
- **Interface de alto nivel con los programadores.**
  - Los programadores de aplicaciones deben disponer de medios sencillos para pedir datos y estar aislados de las complejidades internas de organización y direccionamiento de los archivos.
- **Lenguaje del usuario final.**
  - Un lenguaje de averiguación de alto nivel o un lenguaje para la generación de reportes permitirán que los usuarios finales se vean libres de tener que escribir un programa de aplicación convencional.
- **Controles de integridad.**
  - Siempre que sea posible se recurrirá a chequeos de límites y otros controles para asegurar la exactitud de los datos.
- **Fácil recuperación en caso de fallo.**
  - Recuperación automática sin pérdida de transacciones.
- **Afinación.**
  - La base de datos debe ser afinable para mejorar su desempeño sin exigir la reescritura de

BENEFICIOS

- **Consolidación de archivos.**
- **Independencia archivos-programas.**
- **Seguridad de datos.**
- **Control centralizado de información.**

Cuatro Vistas Diferentes de los Datos

1. VISTA DE LOS PROGRAMADORES DE SISTEMAS O DISEÑADORES DE BASE DE DATOS QUE SE INTERESAN POR EL RENDIMIENTO

ORGANIZACION DEL ALMACENAMIENTO FISICO

REPRESENTADA POR LA DISTRIBUCION FISICA DE LOS DATOS

CONVERSION REALIZADA POR EL SOFTWARE DE ADMINISTRACION DE DATOS

2. VISTA DEL ADMINSTRADOR DE DATOS O DE LOS ANALISTAS DE SISTEMA CON VISTA GLOBAL DE LOS DATOS

ORGANIZACION DE LOS DATOS ALMACENADOS

REPRESENTADA POR EL ESQUEMA

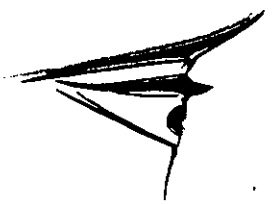
CONVERSION REALIZADA POR EL SOFTWARE DE ADMINISTRACION DE DATOS

3. VISTA DEL PROGRAMADOR DE APLICACIONES Y DE LOS PROGRAMAS DE APLICA ION

ORANIZACION LOGICA PARA LOS PROGRAMADORES

REPRESENTA POR EL SUBESQUEMA

CONVERSION REALIZADA POR LOS PROGRAMAS DE APLICACION O POR EL SOFTWARE DE DIALOGO



4. VISTA DEL USUARIO DE TERMINAL QUE NO ES ESPECIALISTA EN PROCESAMIENTO DE DATOS

ESTRUCTURA DE DATOS PARA SUPRESENTACION EN LAS TERMINALES

REPRESENTADA POR LAS ESPECIFICACIONES DEL DIALOGO HOMBRE-COM-

USUARIOS DEL DBMS

- **1.- Programadores de sistemas. (soporte técnico)**
  - Mantenimiento de software DBMS.
  - Controla el espacio de almacenamiento en disco.
  - Selecciona problemas de software.
- **2.- Personal de operación.**
  - Inicia operación diaria del DBMS.
  - Monitoreo de operación.
  - Monta/desmonta cintas y paquetes de discos.
- **3.- Administración de Base de Datos.**
  - Diseño y definición de la base de datos.
  - Monitoreo de desempeño.
  - Monitoreo para afirmar.
  - Respaldo y recuperación.
  - Seguridad.
- **4.- Programador/analista.**
  - Selecciona problemas de las aplicaciones.
  - Carga datos.
  - Escribe programas.
- **5.- Usuario final.**
  - Analiza los reportes.
  - Corre programas.
  - Define consultas y reportes.
- **6.- Administrador de datos.**
  - Planeación estratégica.
  - Modelo conceptual.
  - Coordinador (comunicación)

RESPONSABILIDADES DEL ADMINISTRADOR DE LA BASE DE DATOS.

- Definir los elementos de datos y las entidades de la empresa.
- Determinar los diferentes nombres que serán usados para referir a los mismos elementos.
- Definir las relaciones entre los elementos de datos.
- Establecer la descripción textual de los elementos de datos.
- Conocer a los departamentos o a los usuarios que serán responsables del manejo de los datos.
- Determinar el uso de los elementos de datos para control planeación es decir, determinar quien esta autorizado para hacer algo.

FUNCIONES DE DESARROLLO DE SISTEMAS DE BASES DE DATOS

- **1.- Planeación estratégica.**
  - Desarrollo de modelo conceptual (administrador de datos).
    - Funciones básicas de organización.
    - Entidades, atributos.
    - Expresarlo en un diccionario de datos.
  - Análisis del modelo.
  - Planeación del proyecto.
- **2.- Desarrollo de la aplicación.**
  - Formulación de requerimientos (Usuario + Analista)
    - Sistemas existentes.
    - Nuevas funciones.
  - Diseño lógico (administrador de base de datos)
    - Base de datos.
    - Requerimientos operacionales (analista).
  - Diseño físico (administrador de base de datos)
    - Base de datos.
    - Programas.
  - Implementación.
    - Conversión de datos.
    - Construcción de programas.
    - Pruebas.
  - Operación + Monitoreo (usuario + operador)
  - Mantenimiento + Adaptación.

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS ABIERTOS

DISEÑO DE BASE DE DATOS

TEMAS COMPLEMENTARIOS

ING. EDUARDO HERNANDEZ OLIVO

JUNIO
1992

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS ABIERTOS

DISEÑO DE BASE DE DATOS

TEMAS COMPLEMENTARIOS

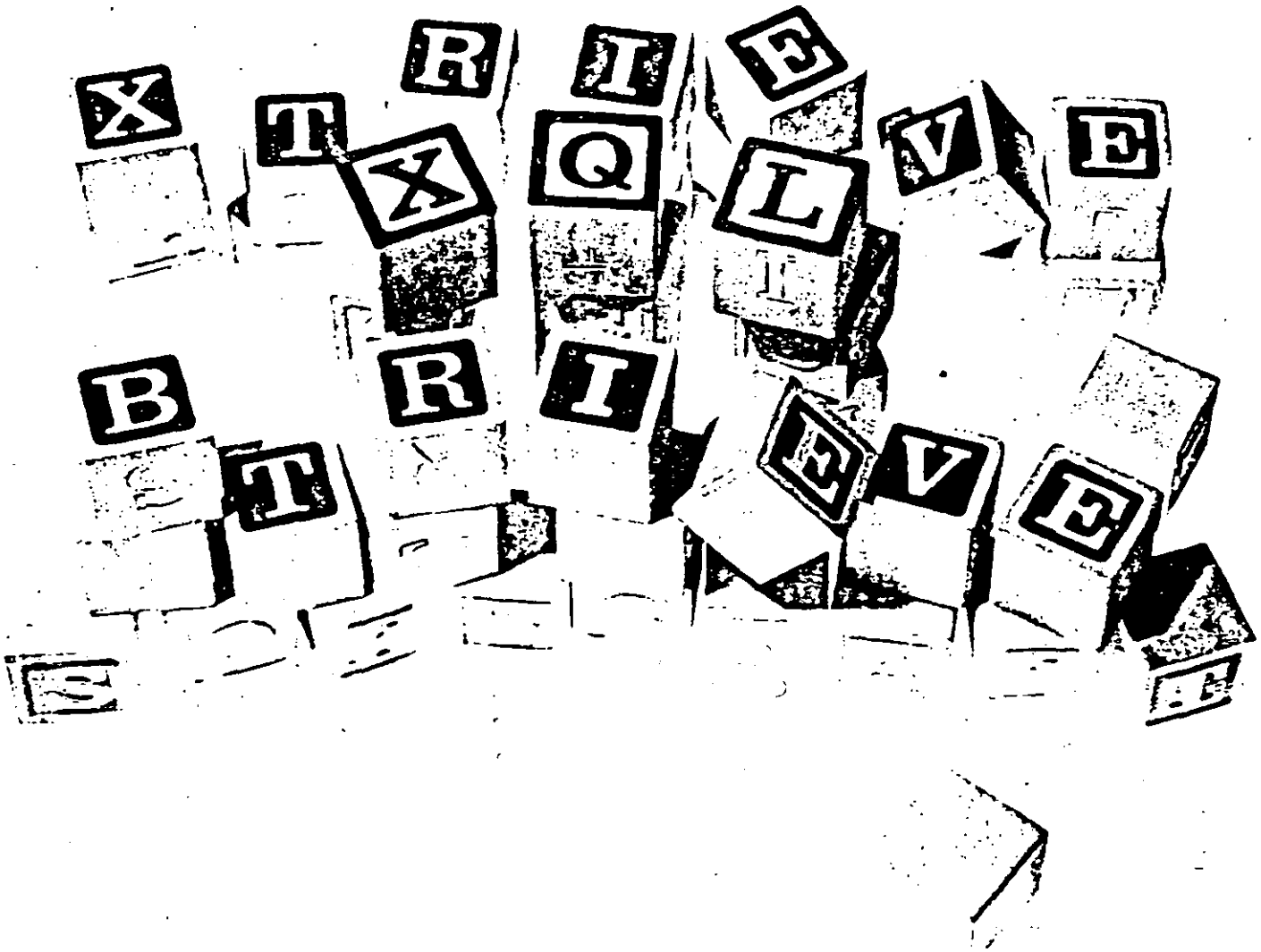
ING. EDUARDO HERNANDEZ OLIVO

JUNIO
1992

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. Tel.: 521-40-20 Apdo. Postal M-2285

A-1
F. L

XQL: The Programmer's SQL



Building Better Applications.

SoftCraft

A Novell Company

XQL—"The Programmer's SQL"

Until now, application developers have been forced to make a choice - use a 4th generation language for convenient database capability or use a real programming language for maximum performance and flexibility. Now you can have both with XQL.

XQL is the relational database management system designed especially for programmers. Imagine being able to access your database with the ease of Structured Query Language (SQL) statements and still having the power to process that data right down to the byte level.

If you write applications then you *need* XQL.

Think about your applications. A large part of your software development effort is probably devoted to managing data stored in files on disk. Hours spent writing lines of code to search and sort data records could have been used to program more important parts of your application.

Why not let XQL do it for you? XQL will increase your programming productivity and let you focus on building better applications.

XQL:

- reduces programming time, *you program less.*
- enhances application capability, *your application does more.*
- improves application performance, *your users don't wait.*

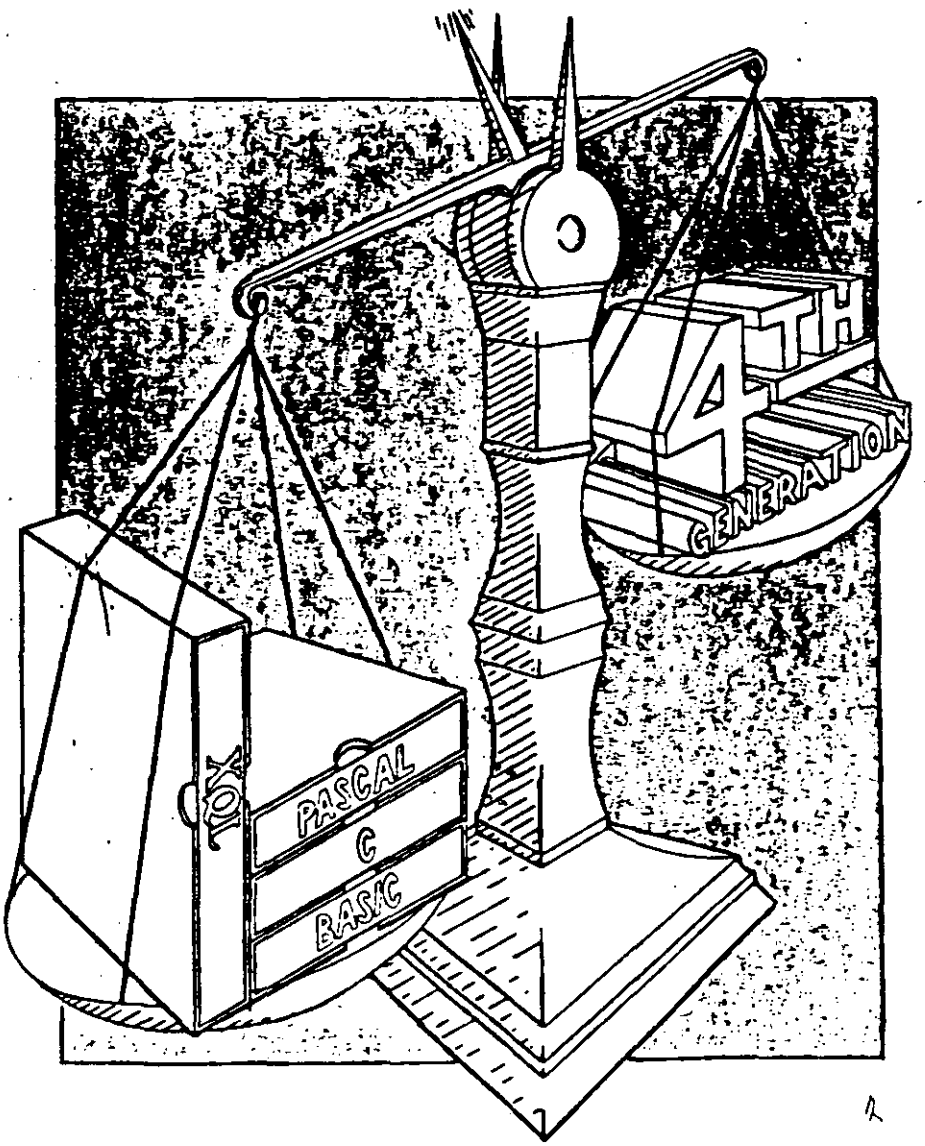
Listen to what Art Nacht of MIN Microcomputer Software Inc. has to say about programming with XQL:

"Using XQL's relational primitives, I was able to reduce 200 lines of an application down to just 25. That alone makes the product a terrific database tool, but I also found XQL's performance to be as good as Btrieve's."

Outstanding results. That's what SoftCraft customers have come to expect from our line of database programming tools.

If you use a 4th generation language to develop applications, then you will *love* XQL.

XQL provides you with all the relational database capabilities you have come to depend on in an SQL environment, and you can use a true programming language. Never again will you have to sacrifice speed and flexibility just to gain the powerful data handling capability of a 4th generation language.



Programming with XQL

XQL lets you perform relational database operations from within a program at two different data access levels:

1) issue complete SQL statements; and,
2) perform relational primitives.

Programming SQL Statements

Perform SQL statements using subroutine calls. Just store each statement in a string variable and pass it to XQL for processing. XQL translates the SQL statement, retrieves the data records you've requested, and returns them sorted in the order you've specified.

Suppose you need a list of all the customers in California with a past due balance. The following SQL statement retrieves those customers:

```
SELECT * FROM customer WHERE state = 'CA'
AND pastdue > 0
```

You make one subroutine call to specify the statement and another to retrieve the records. XQL returns as many records as you want on the call. If necessary, use subsequent calls to get more records. XQL gives you the maximum convenience in processing your data.

```
SQL Example using Quick BASIC

. <initialize data structures here>

OPEN "NUL" AS #1 LEN = 740
FCB.FETCH% = VARPTR(#1)
REM
REM   Login to SQL Manager
REM   Dictionary and data files are in current directory.
REM
CALL XQLLOGIN (STATUS%, USERID$, PASSWORD$,__
     DDPATH$, DATAPATH$, RESERVED$, 0)
IF STATUS% <> 0 THEN
   PRINT "XQLLOGIN FAILED, STATUS:" STATUS%
   CALL QUIT
END IF
REM
REM   Allocate a cursor; this identifies a view
REM
CALL XQLCURSOR (STATUS%, CURSORID%)
IF STATUS% <> 0 THEN
   PRINT "XQLCURSOR FAILED, STATUS:" STATUS%
   CALL QUIT
END IF
STATEMENT$ = __
     +"SELECT Appointment^Date, Appointment^Time,"__
     +"AM/PM, First^Name + Last^Name, Doctor,"__
     +"Procedure^Name FROM"__
     +"patients, appointments procedures"__
     +"WHERE ((ID = Patient^ID) AND"__
     +"(Procedure^Code = Code))"
REM
REM   Compile SQL statement
REM
STATLEN% = LEN (STATEMENT$)
CALL XQLCOMPILE (STATUS%, CURSORID%, STATLEN%, STATEMENT$)
IF STATUS% > 199 THEN
   PRINT "XQLCOMPILE FAILED, STATUS:" STATUS%
   CALL QUIT
END IF
OPT% = 1
WHILE STATUS% = 0
REM
REM   Fetch up to 10 records with single XQLFetch call
REM
     BUFLEN% = 740: RECCOUNT% = 10
     CALL XQLFETCH (STATUS%, CURSORID%, OPT%,__
          BUFLEN%, FCB.FETCH%, RECCOUNT%, 0, 0)
     IF STATUS% > 0 THEN
        PRINT "XQLFETCH FAILED, STATUS:", STATUS%
        CALL QUIT
     END IF

  <process records here>

     OPT% = 2
WEND
CALL QUIT
END

SUB QUIT STATIC
   CALL XQLLOGOUT (STATUS%)
STOP
END SUB
```

```
SQL Example using C

. <initialize data structures here>

/*
 * Login to SQL Manager
 * Dictionary and data files are in current directory.
 */
status = XQLLogin (userid, password, ddpath,
                   datapath, reserved, 0);
if (status) {
  printf ("XQLLogin failed, status: %d\n", status);
  quit 0; return (0);
};
/*
 * Allocate a cursor; this identifies a view
 */
status = XQLCursor (&cursorid);
if (status) {
  printf ("XQLCursor failed, status: %d\n", status);
  quit 0; return (0);
};
/*
 * Build the SQL statement
 */
strcpy (statement,
          "SELECT Appointment^Date, Appointment^Time, ");
stat = statement + strlen (statement);
strcpy (stat, "AM/PM, First^Name + Last^Name, Doctor, ");
stat += strlen (stat);
strcpy (stat, "Procedure^Name FROM ");
stat += strlen (stat);
strcpy (stat, "patients, appointments, procedures ");
stat += strlen (stat);
strcpy (stat, "WHERE ((ID = Patient^ID) AND ");
stat += strlen (stat);
strcpy (stat, "(Procedure^Code = Code))");
stat += strlen (stat);
/*
 * Compile SQL statement
 */
statlen = strlen (statement);
status = XQLCompile (cursorid, &statlen, statement);
if (status) {
  printf ("XQLCompile failed, status: %d\n", status);
  quit 0; return (0);
};
opt = 1;
do {
/*
 * Fetch up to 10 records with single XQLFetch call
 */
     buflen = 740; reccount = 10;
     status = XQLFetch (cursorid, opt, &buflen,
                        &records, &reccount, 0, 0);
     if (status > 99) {
        printf ("XQLFetch failed, status: %d\n", status);
        quit 0; return (0);
     };

  <process records here>

     opt = 2;
  } while (status == 0);
quit 0;
return (0);

quit 0
{
XQLLogout 0;
return (0);
}
```

Programming Relational Primitives

For maximum flexibility, you can bypass the SQL level and perform relational primitive operations directly. You have all the functionality of the relational database model without the constraints of a 4th generation language.

Powerful relational database operations are made easy with straightforward subroutine calls. It's simple to develop a database application using XQL's relational primitives. First, call the subroutine, xNew, to define the file you want to access, followed by xJoin if you want to access multiple files at a time. Then call xField to specify which fields you want returned from each file. These three calls define the logical record which your program sees. No longer must you read multiple data files and extract the important fields. XQL does this for you automatically.

An important relational operation is the subroutine, xRestrict. With it, you establish certain criteria which each record must meet before it is selected and returned to your program. The criteria can be one or mor Boolean conditions connected with "AND" or operators.

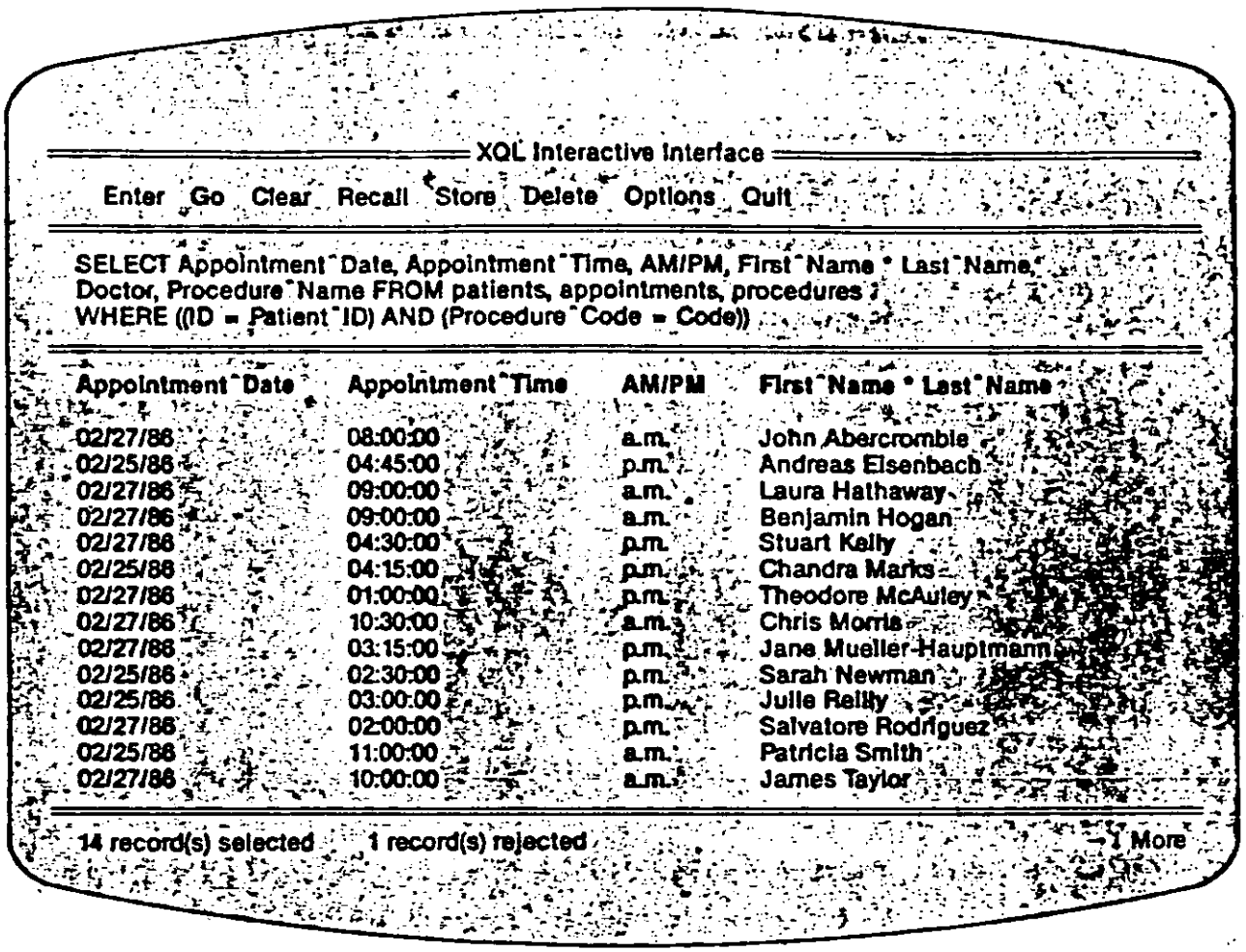
Using these subroutine calls, you have just specified a view (subset) of the database that you want to access. You have defined how each record should be constructed and which records are to be read. Now call the routine xFetch to retrieve the records from this view. Whether you read one record at a time or several, XQL provides a high performance method to retrieve just the data you want. And you can move either forward or backward through your database.

```
XQL Primitives Example using Quick BASIC

  <initialize data structures here>

OPEN "NUL" AS #1 LEN = 740
FCB.FETCH% = VARPTR(#1)
REM
REM   Login to XQLP
REM   Dictionary and data files are in current directory.
REM
Call XLOGIN (STATUS%, USERID$, PASSWORD$, DDPATH$,__
     DATAPATH$, RESERVED$, 0)
IF STATUS% <> 0 THEN
   PRINT "XLOGIN FAILED, STATUS." STATUS%
   CALL QUIT
END IF
REM
REM   Define a new view
REM
CALL XNEW (STATUS%, CURSORID%, FILE1$, OWNER$, -1)
IF STATUS% <> 0 THEN
   PRINT "XNEW FAILED, STATUS:" STATUS%
   CALL QUIT
END IF
REM
REM   Add appointments file to the view
REM
CALL XJOIN (STATUS%, CURSORID%, FILE2$, OWNER$,__
          0, 1, JOINNM11$, 1, JOINNM12$)
IF STATUS% <> 0 THEN
   PRINT "FIRST XJOIN FAILED, STATUS:" STATUS%
   CALL QUIT
END IF
REM
REM   Add procedures file to the view
REM
CALL XJOIN (STATUS%, CURSORID%, FILE3$, OWNER$,__
          0, 1, JOINNM21$, 1, JOINNM22$)
IF STATUS% <> 0 THEN
   PRINT "SECOND XJOIN FAILED, STATUS:" STATUS%
   CALL QUIT
END IF
REM
REM   Define 5 fields for the view
REM
FCOUNT% = 5
FLDNAMES$ = "Appointment^Date"+CHR$(0)__
          +"Appointment^Time"+CHR$(0)__
          +"AM/PM"+CHR$(0)__
          +"Doctor"+CHR$(0)__
          +"Procedure^Name"+CHR$(0)
CALL XFIELD (STATUS%, CURSORID%, 0, 0__
          FCOUNT%, FLDNAMES$)
IF STATUS% <> 0 THEN
   PRINT "XFIELD FAILED, STATUS:" STATUS%
   CALL QUIT
END IF
OPT% = 1
REM
REM   Fetch up to 10 records with single xFetch call
REM
WHILE STATUS% = 0
   BUFLEN% = 740: RECCOUNT% = 10: REJCOUNT% = -1
   CALL XFETCH (STATUS%, CURSORID%, BUFLEN%,__
        OPT%, RECCOUNT%, REJCOUNT%, FCB.FETCH%)
   IF STATUS% = 0 OR STATUS% = 9 THEN

      <process records here>

      OPT% = 2
   ELSE
      PRINT "XFETCH FAILED, STATUS:", STATUS%
      CALL QUIT
   END IF
WEND
CALL QUIT
END

SUB QUIT STATIC
   CALL XRESET (STATUS%, RESERVED$)
   CALL XLOGOUT (STATUS%)
   STOP
END SUB
```

```
XQL Primitives Example using C

  <initialize data structures here>

/*
 * Login to XQLP
 * Dictionary and data files are in current directory.
 */
status = xLogin (userid, password, ddpath,
                 datapath, reserved, 0);
if (status) {
   printf ("xLogin failed, status: %d\n", status);
   quit (); return (0);
};
/*
 * Define a new view
 */
status = xNew (&cursorid, file1, owner, -1);
if (status) {
   printf ("xNew failed, status: %d\n", status);
   quit (); return (0);
};
/*
 * Add appointments file to the view
 */
status = xJoin (cursorid, file2, owner,
                0, 1, joinnm11, 1, joinnm12);
if (status){
    printf ("first xJoin failed, status: %d\n", status);
    quit (); return (0);
};
/*
 * Add procedures file to the view
 */
status = xJoin (cursorid, file3, owner,
                0, 1, joinnm21,1, joinnm22);
                if (status) {
                    printf ("second xJoin failed, status: %\n", status);
                    quit (), return (0);
};
/*
 * Define 5 fields for the view
 */
fcount = 5;
b    = fldnames;
strcpy (b, "Appointment^Date"); b += strlen(b) + 1;
strcpy (b, "Appointment^Time"); b += strlen(b) + 1;
strcpy (b, "AM/PM");        b += strlen(b) + 1;
strcpy (b, "Doctor");       b += strlen(b) + 1;
strcpy (b, "Procedure^Name");
status = xField (cursorid, 0, 0, &fcount, fldnames);
if (status) {
    printf ("xField failed, status: %d\n", status);
    quit (); return (0);
};
opt = 1;
/*
 * Fetch up to 10 records with single xFetch call
 */
do {
    buflen = 740; reccount = 10; rejcount = -1;
    status = xFetch (cursorid, &buflen, opt,
                     &reccount, &rejcount, records);
    if (status == 0 || status == 9) {

      <process records here>

       opt = 2;
    } else {
       printf ("xFetch failed, status: %d\n", status);
       quit (); return (0);
    };
}
while (status == 0);
quit ();
return (0);
}

quit ()
{
   xReset (reserved);
   xLogout ();
   return (0);
}
```

Interactive SQL

XQL even includes its own interactive SQL editor so you can test each statement before embedding it into your application. As a debugging tool, it allows you to view the actual data that will be returned to your program. And it's perfect for performing ad-hoc queries. Create a library of queries by storing SQL statements on disk and recalling them later for instant data lookup.



XQL: For Productivity and Performance

When you purchase XQL you're buying more than a piece of software. You're buying SoftCraft's years of expertise in database management and our commitment to quality and service.

XQL is the latest entry in SoftCraft's continuing effort to bring sophisticated data management tools to PC programmers. By purchasing XQL you put these powerful features into each and every XQL program you write:

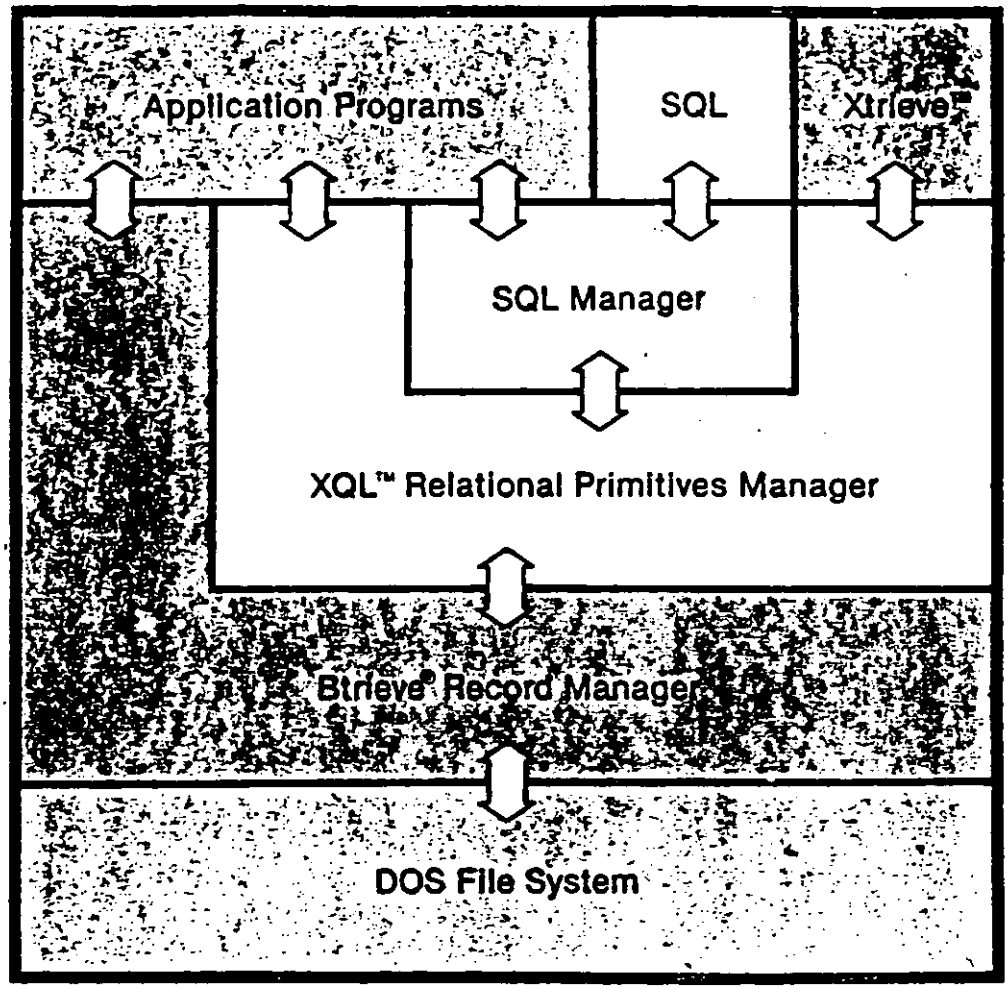
- a single version supports both single and multi-user Btrieve
  LANs and DOS multi-tasking
- data independence
  request data by field name
- data integrity
  transaction control and automatic index recovery
- automatic query optimization
  b-tree indexes
- data validation on input
  ranges, character lists, and value lists
- password security
  read/write access to the field level
- no runtime royalties

Btrieve, the programmer's choice for safe and efficient file management, is the heart of XQL. This means XQL offers all the performance and reliability programmers have come to expect from Btrieve, incuding LAN support, fault tolerance, comprehensive documentation and expert technical help.

In fact, XQL is part of a comprehensive line of quality database management tools available from SoftCraft. In addition to Btrieve and XQL, SoftCraft offers Xtrieve™, an interactive, menu driven query system and report writer that gives your users access to Btrieve files without having to write a program. Xtrieve's power and flexibility make it easy to customize for your environment and thus an ideal product for you to resell to your customers.

The following diagram shows how XQL works with Btrieve and Xtrieve to give you the best possible database handling capability.

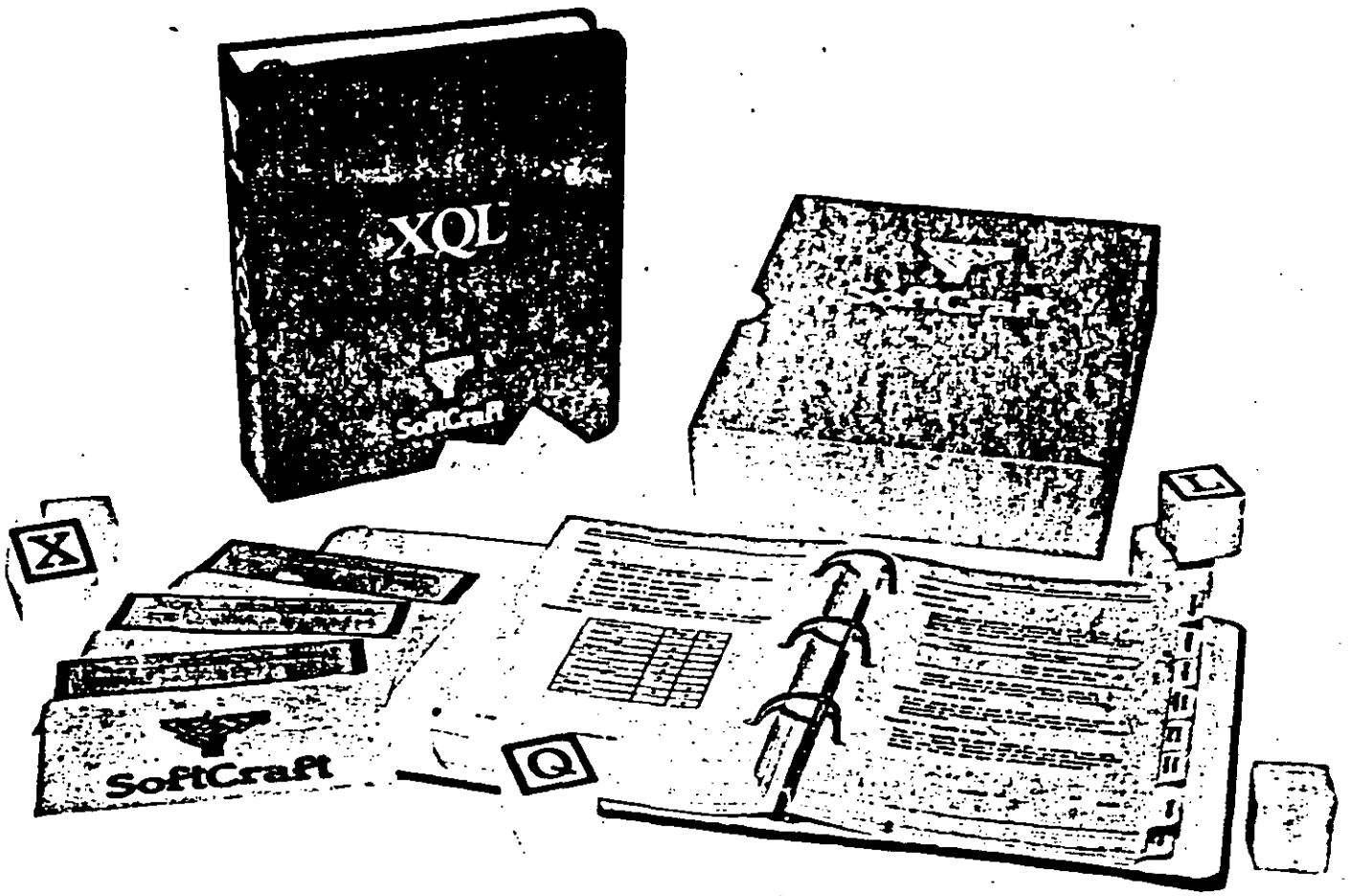
SoftCraft Product Architecture



Using SoftCraft's complete product line, an application program can make subroutine calls directly to:

1) the SQL Manager;
2) the XQL Relational Primitives Manager; or,
3) the Btrieve Record Manager.

The SQL Manager translates SQL statements into relational primitives and makes calls to the XQL Relational Primitives Manager. The Primitives Manager, in turn, translates primitives into Btrieve operations and calls the Btrieve Record Manager. Xtrieve operates as an application program, issuing relational primitives directly to the Primitives Manager.



XQL Relational Primitives

XQL provides the following data definition and data manipulation primitive operators. The data dictionary primitives, security and administrative primitives, and miscellaneous primitives are not included in this list.

xNew starts a new view definition and defines the file for the view

xFree releases all resources for an active view definition

xField defines the field list for a view

xMovefld changes the position of a field within a view

xDescribe returns the description for a view, based on the specified subfunction

xFetch retrieves data for the specified view

Compute defines a computed field for a view

xOrder defines the sort order for a view

xRestrict defines the restriction for the records in the view

xJoin defines the fields used to join files

xInsert inserts records into the file(s), of an active view

xUpdate updates records, in the file(s), returned by the last xFetch

xUpdall updates all records, for the file(s), contained in the view

xRemove removes records, from the file(s), returned by the last xFetch

xRemall removes all records, from the file(s), included in the view

xReset releases all active view definitions for a session

xTrans performs the transaction related operation which is specified in the subfunction

xStore stores an active view definition in the current dictionary

xRecall recalls a stored view from the current dictionary

XQL Keywords

XQL supports the following keywords. Keywords which are marked with an asterisk are XQL extensions to SQL. Keywords in parentheses are acceptable abbreviations of the preceding keyword.

ACCELERATED*	CREATE	IN	NOTE*	START*
ADD	CREATETAB*	INDEX*	NULL	STRINGNUL
ALL	DATAPATH*	INSERT	NUMERIC	SUM
ALTER	DATE*	INTEGER (INT)	ON	TABLE
AND	DDPATH*	INTO	OPENMODE*	TIME*
AS	DECIMAL (DEC)	IS	OR	TO
ASC	DECIMALNULL*	LOGICAL	ORDER	TRANSACTION*
AVG	DEFAULT*	LOGIN*	OWNER*	UNIQUE
BEGINS*	DELETE	LSTRING*	RANGE*	UPDATE
BETWEEN	DESC	LVAR*	READONLY*	USER
BFLOAT*	DICTIONARY*	MASK*	REPLACE*	USING*
BINARYNULL*	DROP	MAX	REVOKE	VALUES
BLANK*	EXCLUSIVE*	MIN	ROLLBACK	VERIFY*
BY	FLOAT	MOD*	SECURITY*	VIEW
CASE*	FROM	MODIFY	SEG*	WHERE
CHARACTER (CHAR)	GRANT	MONEY*	SELECT	WITH
COMMIT	GROUP	NORMAL*	SET	WORK
CONTAINS*	HAVING	NOT	SMALLINT	ZSTRING*
COUNT				

Technical Data

Records per file	No limit
Maximum file size	4 gigabytes
Maximum record size	4K
Maximum field size	255
Maximum indexes per file	24
Index Attributes	duplicate/unique, modifiable/nonmodifiable, ascending/descending, case sensitive/insensitive, null, segmented
Maximum fields per view	No limit
Maximum rows per view	No limit
Maximum joined files per view	8
Maximum active views	No limit
Maximum files per database	No limit
Hardware Supported	IBM PC, XT, & AT; and compatibles
Data type	string, signed integer, floating point, date, time, decimal, money, BASIC floating point, numeric strings, Boolean, variable length
Restrict operators	begins with, contains, does not contain, equal to, not equal to, greater than, less than, greater than or equal, less than or equal, and, or
Language interfaces	MS-C, Lattice C, MS-BASIC, Quick BASIC, Turbo Pascal, documentation provided for user written interfaces
Requires	MS-DOS 3.X, Btrieve 4.1X

Runtime Size

Primitives Manager	adjustable from 90K to 180K
SQL Manager	adjustable from 80K to 130K

(requires Primitives Manager be loaded)

SoftCraft
A Novell Company

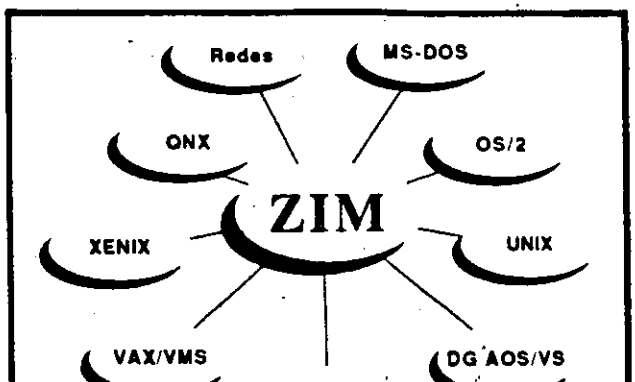
P.O. Box 9802 #917
Austin, TX 78766

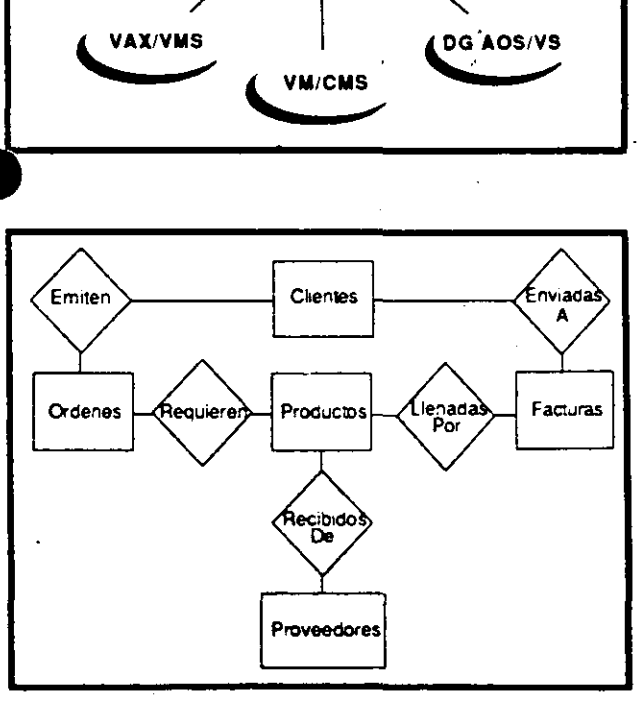
Telephone: 512/346-8380
Telex: 358 200

ZIM

INNOVACION TECNOLOGICA EN BASE DE DATOS

ZIM el sistema de desarrollo de aplicaciones más avanzado, es un administrador de bases de datos y lenguaje de cuarta generación (DBMS/4GL) totalmente integrado. **ZIM** esta basado en el modelo de datos de Entidades y Relaciones (E-R) y es completamente transportable, sin cambios en el código, entre los siguientes sistemas operativos: MS-DOS, redes de MS-DOS, redes Novell, OS/2, QNX, XENIX, UNIX, VMS, AOS/VS y VM/CMS.





Los beneficios globales que se obtienen con **ZIM**, tales como mejoras en la calidad del diseño, en la programación, implantación y mantenimiento, y en el desempeño de las empresas gracias a la mayor disponibilidad de información útil para la toma de decisiones, lo convierten en una herramienta imprescindible en los sistemas de cómputo de todas las empresas.

DBMS

Como sistema administrador de bases de datos, **ZIM** coincide con la idea de que la información es un recurso de enorme importancia en las organizaciones. **ZIM** facilita la administración de la información generada por computadora, facilita la compartición de los datos entre distintos usuarios, provee información más oportuna, minimiza la redundancia de los datos, reduce el costo de mantenimiento y mejora la consistencia, integridad y seguridad de la información.

4GL

ZIM es un lenguaje flexible y poderoso que permite una gran productividad de programación y reducido mantenimiento. Como 4GL, **ZIM** combina todas las facilidades necesarias para el desarrollo de aplicaciones en un solo lenguaje uniforme: diccionario de datos, consulta y actualización de la base de datos, manipulación de formas, escritura de reportes, construcciones para programar estructuradamente importación/exportación de datos y depuración.

MODELO E-R

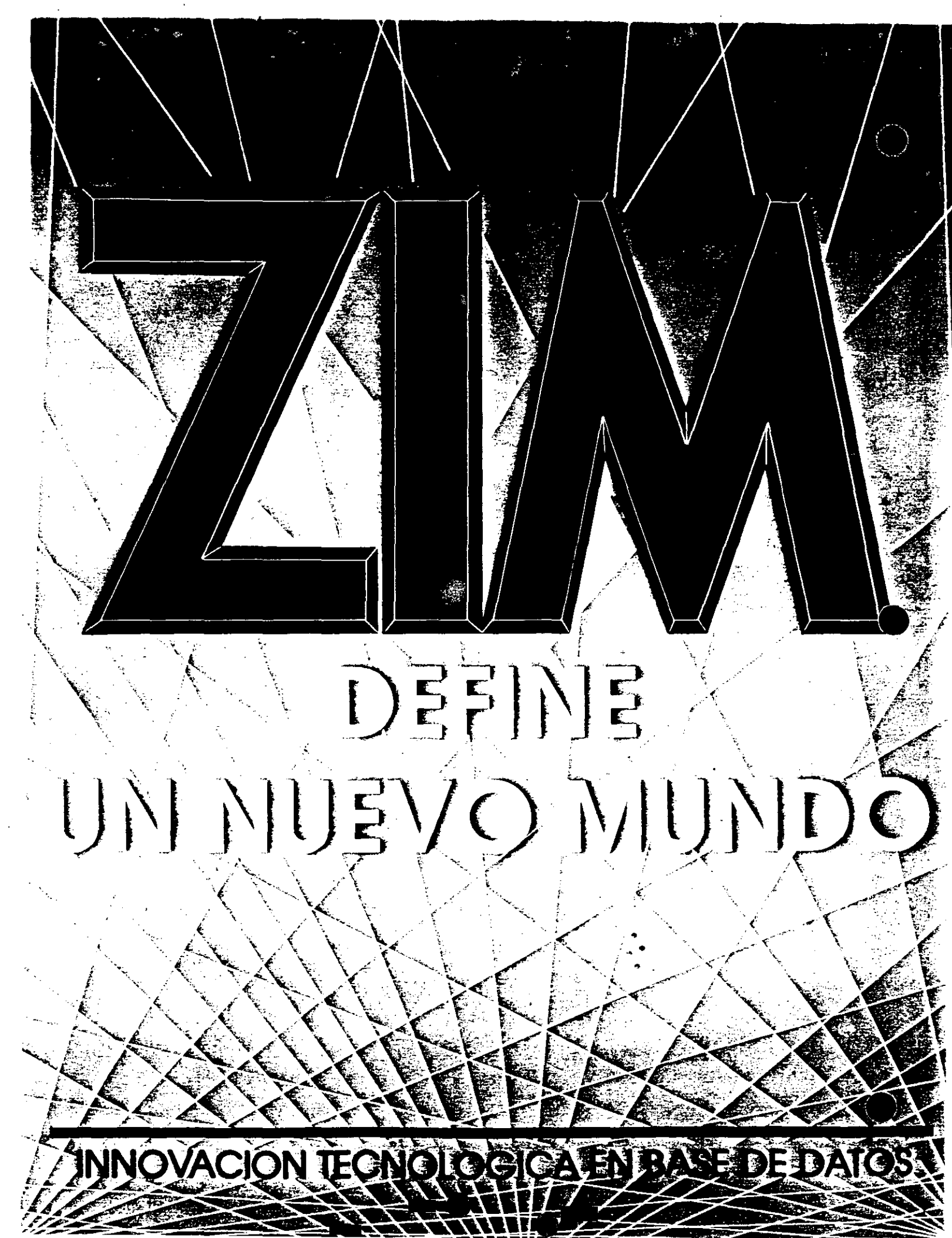
ZIM está basado en el modelo de datos de Entidades y Relaciones. Este modelo es una metodología que incorpora información semántica importante del mundo real, a través del uso de una técnica especial de diagramas para el diseño y descripción de las bases de datos. Con esta técnica, la base de datos resultante elimina código redundante y repetitivo dentro de la aplicación. Asimismo, se evitan problemas de duplicación de datos, actualización y mantenimiento.

100% TRANSPORTABLE

ZIM asegura su inversión en el desarrollo de sus sistemas y programas de aplicación al ser 100% transportable entre los principales sistemas operativos. Es decir, las aplicaciones realizadas en **ZIM** no requieren cambios en el código cuando se transfieren entre sistemas operativos. Con ello, los departamentos de sistemas tienen la opción de desarrollar sus aplicaciones en PCs, reduciendo la carga en los caros y a menudo escasos recursos en minis y mainframes.

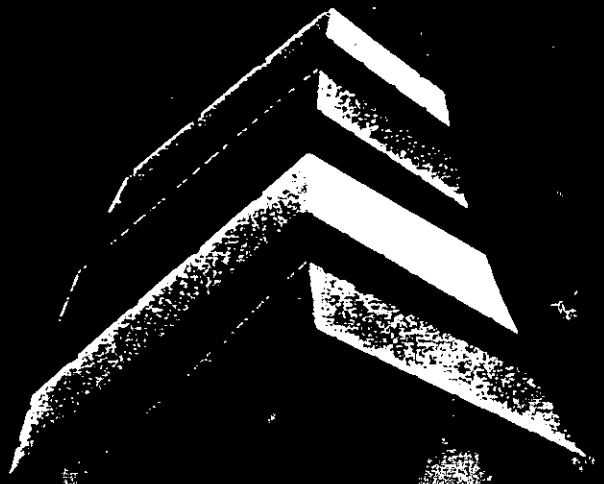
*M.R. de Sterling Software
Zanthe Systems Division y los respectivos fabricantes

Programación en Tiempo Real, S.A. de C.V. 5 de febrero No. 662-3 Col. Alamos 03400 México, D.F. Tel: 519 03 36 Fax: 530 97 02


ZIM
DEFINE
UN NUEVO MUNDO
INNOVACION TECNOLOGICA EN BASE DE DATOS

Sistema MAPPER

El Sistema Flexible que Permite Mayor Creatividad al Usuario Final



Rompa la Barrera del Lenguaje entre su Personal y las Computadoras

El Sistema MAPPER® ayuda a su personal a diseñar sus propias soluciones para superar sus propios retos profesionales. Pone la creatividad de la computadora al alcance de toda su organización y de la gente que más la necesita.

Al contar con sus propios sistemas de información, los profesionales de su empresa pueden trabajar más y con mayor eficiencia. Adaptarán las soluciones de computadora a su estilo personal de trabajo, con base en su experiencia única. La información oportuna y relevante les ayudará a tomar mejores decisiones reforzando la competitividad de su empresa a todos los niveles.

El Sistema MAPPER permite a su personal usar directamente las computadoras sin la participación del personal de Informática. Proporciona a los usuarios finales las herramientas necesarias para desarrollar sus propios sistemas sin tener que familiarizarse con la terminología propia de la computación.

"El Sistema MAPPER permite a los usuarios transferir ideas e información instantánea y directamente al sistema y a sus colaboradores. Y, además, lo hace sin necesidad de recurrir a la programación y planeación convencionales de sistemas."

-Lou Schlueter, *analista de computación.*

El sistema MAPPER:

- *Ayuda a quienes no tienen experiencia en programación a crear soluciones de acuerdo a sus necesidades*
- *Provee a la gerencia con información que facilita el control y la planeación estratégica*
- *Reduce las cargas de aplicaciones pendientes*
- *Maneja con mayor eficiencia los recursos del departamento de Informática.*

Conexión de Sistemas para Lograr un Ambiente de Información Unificado

La mayoría de los usuarios finales realizan funciones administrativas en las áreas de planeación estratégica y de mercados, investigación, administración de finanzas y análisis de negocios. Pocos de estos usuarios tienen una formación formal en Informática.

Estas actividades se realizan a nivel departamental o corporativo. Los Sistemas MAPPER pueden conectarse a otros sistemas, ayudándole a satisfacer las necesidades de información de todo tipo de estructura organizacional, desde profesionales trabajando en forma independiente hasta tomadores de decisiones a nivel corporativo. Con esto se logra un uso más productivo e integral de la información sin riesgo de afectar la base de datos corporativa y sin complicaciones para el respaldo, la recuperación o la seguridad de la información.

Ampliación de los Límites del Equipo

Todos los Sistemas MAPPER son totalmente compatibles. Se puede recurrir a una mayor potencia de cómputo y rendimiento sin cambiar el software. Pueden integrarse nuevas soluciones con el mismo equipo y software. De esta manera se protege su inversión en sistemas de información sin afectar su crecimiento.

Ponga el Poder de la Computadora donde lo Necesite

Con MAPPER la información está disponible para quien la necesita. Usted satisface las necesidades de información a cualquier nivel: organizacional, departamental y personal. El acceso más fácil a la información estratégica le ayuda a:

- Manejar el cambio en su ambiente particular de negocios
- Tomar decisiones más efectivas y oportunas
- Manejar y controlar sus recursos de información
- Aprovechar al máximo el talento de los profesionales que no son programadores

Con MAPPER, Unisys está llevando estos beneficios al alcance de más de un millón de usuarios en más de 8,000 instalaciones. Podemos hacer lo mismo para usted. Para descubrirlo cómo acuda con el Representante Unisys mas cercano.

"El Sistema MAPPER crea adicción. Es fácil comenzar a usarlo. Y a medida que los nuevos usuarios se familiarizan con sus funciones, rapidamente lo usarán cada vez más en su trabajo. Convirtiendose en estusiastas misioneros del enfoque del Sistema MAPPER."

-James M n

Rompa la Barrera del Lenguaje entre su Personal y las Computadoras

El Sistema MAPPER® ayuda a su personal a diseñar sus propias soluciones para superar sus propios retos profesionales. Pone la creatividad de la computadora al alcance de toda su organización y de la gente que más la necesita.

Al contar con sus propios sistemas de información, los profesionales de su empresa pueden trabajar más y con mayor eficiencia. Adaptarán las soluciones de computadora a su estilo personal de trabajo, con base en su experiencia única. La información oportuna y relevante les ayudará a tomar mejores decisiones reforzando la competitividad de su empresa a todos los niveles.

El Sistema MAPPER permite a su personal usar directamente las computadoras sin la participación del personal de Informática. Proporciona a los usuarios finales las herramientas necesarias para desarrollar sus propios sistemas sin tener que familiarizarse con la terminología propia de la computación.

"El Sistema MAPPER permite a los usuarios transferir ideas e información instantánea y directamente al sistema y a sus colaboradores. Y, además, lo hace sin necesidad de recurrir a la programación y planeación convencionales de sistemas."

-Lou Schlueter, *analista de computación.*

LENGUAJES DE 4a. GENERACION

CLASIFICACION

1. – SIMPLES GENERADORES DE PROGRAMAS EN COBOL.
2. – LENGUAJES GENERADORES DE PROGRAMAS EN COBOL CON APOYO DE SISTEMAS ADMI – NISTRADORES DE BASES DE DATOS.
3. – L4G VERDADEROS QUE ADEMAS DE LAS CA – RACTERISTICAS ANTERIORES, UTILIZAN EN GRAN PARTE EL SOFTWARE DEL PROVEEDOR DE HARDWARE, COMO SISTEMAS DE COMU – NICACIONES, RESPALDO PARA PROTECCION ASI COMO VARIAS UTILERIAS.

LENGUAJES DE 4a. GENERACION

CARACTERISTICAS

- FACIL APRENDIZAJE
- ALTA PRODUCTIVIDAD 10:1
- ESTAN DISE\ADOS PARA OPERAR EN LINEA
- INTERFAZ CON UNA DBMS
- DICCIONARIO DE DATOS
- INDEPENDENCIA CON S.O.
- GENERAN CODIGO ESTRUCTURADO
- FACILMENTE MODIFICABLES
- EL USUARIO FINAL CREA SUS PROPIAS APLI-CACIONES, ASI COMO SUS REPORTES.

LENGUAJES DE 4a. GENERACION

E J E M P L O S

- MAPPER
- LINC
- COGEN
- ZIM
- POWERHOUSE
- ADR / IDEAL
- ETC.

ESTUDIOS DE CASOS

SISTEMAS RELACIONALES

Sistema R
SQL / Data System
Database 2
Oracle
Ingres

SISTEMAS DE RED

Total
IDMS

SISTEMAS JERARQUICOS

IMS
Sistema 2000

SISTEMAS DE BD PARA MICROS

dBASE III +
Dataflex
Visicalc
Lotus 1-2-3

las. En Lynch (1983)
ıcciones para incluir
ın de las transaccio-
latos de CAD. Tam-
ıtos que no requiere
ıjamiento de la limi-

le conocimientos y su
esenta un grupo de
ofrece un tratamiento
cia artificial. Wieder-
ıe se aplican técnicas
et al. (1983), Berns-
npleo del modelo en
z con usuarios. Korth
a extender los mode-
on el usuario mejora-
lenguaje de consulta
mientos de lenguajes
de conjuntos y tipos
83), Roth et al. (1985)

Estudios de casos

En este capítulo se analizarán algunos sistemas de base de datos selectos que, o bien se distribuyen en forma comercial, o son sistemas experimentales que han tenido un impacto importante. No se proporcionarán los detalles completos de los sistemas elegidos. Más bien, se harán resaltar las características más importantes de los sistemas.

Dado el número de sistemas que pueden adquirirse en el mercado, y la gran cantidad de sistemas que se liberaron mientras se escribía este libro, la descripción que sigue dista mucho de ser completa. Los sistemas se eligieron para ilustrar los conceptos presentados en este texto. En consecuencia, el hecho de haber elegido un sistema no implica que se recomiende el producto.

15.1 Sistemas relacionales

El término *relacional* se ha aplicado a un gran número de sistemas, incluso a algunos que, a pesar de utilizar *tablas*, no siguen el espíritu de la definición original del modelo de datos relacional. Para que un sistema pueda considerarse relacional, es necesario que cumpla con las siguientes condiciones:

- Los datos se almacenan en tablas.
- El usuario no puede distinguir apuntadores o ligas.
- El lenguaje de consulta es completo en cuanto a las relaciones.
- Las consultas pueden expresarse sin utilizar iteraciones o proposiciones recursivas.

Los sistemas cuyas estructuras de datos básicas son tablas, pero que imponen restricciones en cuanto a las uniones que se permiten, no se consideran relacionales. Para este tipo de sistemas se utilizará el término *tabulares*.

15.1.1 Sistema R

El proyecto de investigación del sistema R se inició en el IBM San José Research Laboratory en 1974. El objetivo del proyecto era demostrar la aplicación práctica del modelo de datos relacional, que en ese entonces se acababa de proponer. Para ello, era necesario demostrar que el modelo relacional era apropiado para una interfaz con el usuario, y descubrir formas de implantar un sistema relacional que procesara de manera eficiente las consultas.

El éxito de estos trabajos ha quedado demostrado tanto por los productos comerciales que se derivaron de las investigaciones sobre el Sistema R como por el hecho de que dichas investigaciones establecieron un buen número de conceptos fundamentales. El proyecto Sistema R se terminó en 1979. Poco tiempo después IBM anunció el producto de base de datos *SQL/Data System* (véase la sección 15.1.2) y formó el proyecto de investigación de la base de datos distribuida R* para extender las investigaciones del Sistema R a las bases de datos distribuidas.

Entre las contribuciones más importantes del proyecto Sistema R están:

- El lenguaje de consulta SQL.
- La compilación y optimización de consultas.
- La integración de un lenguaje relacional y un lenguaje de programación convencional.
- La serialibilidad y el protocolo de candados de dos fases.
- El protocolo de granularidad múltiple para candados.

El SQL se describió en forma detallada en el capítulo 3, y los candados de granularidad múltiple se presentaron en el capítulo 11.

Estructura general del sistema

La arquitectura interna del Sistema R consta de dos componentes principales:

- Sistema de almacenamiento relacional (RSS, *Relational Storage System*), que se encarga de manejar la memoria (almacenamiento en disco y memoria principal), de la recuperación de caídas, y del control de concurrencia.
- Sistema de datos relacional (RDS, *Relational Data System*), que se encarga de vistas, autorizaciones e integridad.

El RSS permite tener acceso registro por registro a las relaciones del modelo lógico. El acceso a niveles más altos se logra por medio de la interfaz RDS. Ésta acepta expresiones en SQL.

Sobre la base del RDS puede construirse un número arbitrario de interfaces con la base de datos compartida del Sistema R. La que se utiliza con más frecuencia es la interfaz amable con el usuario (UFI, en inglés, User-Friendly Interface). La UFI es un programa que acepta proposiciones en SQL del usuario, las turna

al RDS y despliega los resultados. Pueden escribirse programas de aplicacic que utilicen directamente el RDS.

Compilación de consultas

Los usuarios comerciales de sistemas de base de datos cuentan con una serie de consultas que se ejecutan en forma periódica para generar reportes. Para evitar la necesidad de elegir una estrategia de procesamiento de consultas cada vez que se hace una de éstas, el Sistema R permite la *precompilación* de consultas. La precompilación consiste del análisis sintáctico de la consulta y de la elección de una estrategia completa para procesarla. El resultado de la precompilación se conoce como *plan*. Los planes se almacenan en la base de datos del Sistema R y se leen cada vez que se ejecuta la consulta.

Dado que un plan incluye detalles tales como la elección de estrategias de unión y la utilización de índices específicos, es posible que un plan deje de ser válido. Por ejemplo, podría darse el caso de que se borrara un índice utilizado por el plan. En estas situaciones es necesario volver a ejecutar la compilación de la consulta antes de que pueda ejecutarse ésta.

Lenguajes huésped

Cuando se presentaron los modelos de datos de red y jerárquico en los capítulos 4 y 5, se vio la necesidad de incrustar las solicitudes de acceso al sistema de base de datos dentro de un lenguaje de programación huésped. El modelo relacional no requiere de un lenguaje de consulta que incluya iteraciones, recursión o apuntadores, por lo que no es necesario incrustar los lenguajes relacionales que se presentaron en el capítulo 3 dentro de un lenguaje huésped. No obstante, la posibilidad de hacerlo muchas veces es útil en las aplicaciones comerciales de base de datos. Por ejemplo, un programador podría querer desarrollar una interfaz de aplicación especial con la base de datos para un grupo específico de usuarios. Para facilitar el desarrollo de tales aplicaciones, el Sistema R incluye al *SQL incrustado* (*embedded SQL*), que es una forma con ligeras modificaciones del SQL y que puede utilizarse dentro de un lenguaje de programación huésped. El Sistema R permite dos lenguajes huésped: PL/1 y Cobol. En el análisis que sigue se hablará sólo del PL/1.

Los problemas principales que implica la inclusión de los conceptos de lenguaje del SQL en el PL/1 surgen del hecho de que éste opera sobre registros, mientras que el SQL opera sobre conjuntos de registros (relaciones). Por tanto, se necesita un mecanismo para presentar al programa en PL/1 el resultado de una consulta en SQL (es decir, una relación) tupla por tupla (o sea, registro por registro). Para cada una de las relaciones que va a procesar el programa en PL/1 se define un *cursor*. Este es un apuntador a una tupla y tiene una función similar a la de los apuntadores de actualidad del lenguaje de manipulación de datos DBTG. A continuación se resume el método de acceso normal de un programa en PL/1 a la base de datos:

- Se llama a un procedimiento que hace que se ejecute la consulta en SQL.

- Se abre un *cursor* para la relación que resulta de la ejecución de la consulta. Este cursor se utiliza para procesar una por una las tuplas de la relación resultante.
- Se hace una llamada de *búsqueda* (*fetch*) para obtener la "siguiente" tupla. La búsqueda obtiene la primera tupla de una relación asociada con un cursor que se acabe de abrir. Las siguientes llamadas de búsqueda avanzan al cursor hasta la siguiente tupla de la relación y leen la tupla a la que apunta el cursor. La tupla así obtenida se coloca en un registro de PL/1 para que el programa en PL/1 la pueda manipular.
- Se repiten las llamadas de búsqueda hasta que se hayan procesado todas las tuplas. Cuando esto sucede, la llamada de búsqueda produce un código especial.

Un programa real es ligeramente más complejo, debido a que:

- Es necesario que un preprocesador traduzca las proposiciones en SQL antes de que el compilador de PL/1 compile el programa. Se requieren comandos especiales para el preprocesador.
- Debe incluirse un procedimiento para transferir los valores de una tupla a variables de PL/1.
- Es posible que el tamaño de las tuplas que obtenga la búsqueda dependa de entradas que se hacen al programa. Esto sucede, por ejemplo, cuando el usuario introduce una consulta en SQL al programa mientras se está ejecutando.

En un programa real en PL/1 con SQL incrustado, aparece una cláusula adicional en las proposiciones en SQL. Ésta, la cláusula **into** ("a"), especifica las variables de PL/1 en las que se debe colocar el valor de cada uno de los atributos de la cláusula **select**. Para distinguir los nombres de variables de PL/1 de los nombres de relaciones y atributos, se les agrega un signo de dólar como prefijo siempre que aparecen en una proposición de SQL. Las variables de PL/1 pueden utilizarse también para efectuar comparaciones en la cláusula **where**.

Por ejemplo, considérese la consulta "encontrar los nombres de los clientes que tienen más de x dólares en alguna cuenta, y las ciudades donde viven", donde x es una variable del programa en PL/1. Supóngase que se utilizan las variables de PL/1 a y b para guardar una tupla del resultado. La consulta se escribe así:

```
select nombre-cliente, ciudad-cliente
   into $a, $b
   from depósito, cliente
   where depósito.nombre-cliente = cliente.nombre-cliente
         y depósito.saldo > $x
```

Los comandos para el preprocesador del Sistema R comienzan con el signo dólar. Existen cuatro comandos:

- **$declare.** Este comando es idéntico al comando *declare* (declarar) del PL/1 y sirve para definir a las variables de PL/1 que se utilizan también en consultas en SQL.
- **$let** <nombre-cursor> **be** <proposición en SQL> . Este comando asocia a la proposición en SQL con el nombre del cursor. El procesador almacena esta asociación para utilizarla después. El comando **$let** *no* hace que se evalúe la consulta.
- **$open** <nombre-cursor> ..Este comando hace que el preprocesador genere una llamada de PL/1 a un procedimiento que evalúa la consulta en SQL que está asociada al cursor. La asociación entre un nombre de cursor y una consulta en SQL debe haberse definido con anterioridad por medio de un comando **$let.** En este punto se produce una versión precompilada de la proposición en SQL. Cabe hacer notar que la proposición **into** no influye en este punto sobre el procesamiento de la consulta en SQL.
- **$fetch** <nombre-cursor> . Este comando hace que se coloquen los valores de la siguiente tupla de la relación asociada al cursor que está abierto en las variables de PL/1 que se listan en la cláusula **into.**

Para ilustrar los comandos anteriores se presentará un programa que imprime los resultados del ejemplo anterior, "encontrar los nombres de los clientes que tienen más de x dólares en alguna cuenta, y las ciudades donde viven":

```
$declare a char (20);
$declare b char (20);
$declare x binary fixed (15);
declare rc binary fixed (15);
$let c be select nombre-cliente, ciudad-cliente
          into $a, $b
          from depósito, cliente
          where depósito.nombre-cliente = cliente.nombre-cliente
                y depósito.saldo > $x
$open c;
rc = 0;
do while rc = 0;
      $fetch c;
      put skip list (a, b);
end
```

El procesador convierte el programa en uno susceptible de ser compilado por el compilador de PL/1. También precompila la consulta en SQL y almacena el plan correspondiente en la base de datos.

El ejemplo anterior se simplificó debido a que el programador de aplicaciones sabía que la consulta en SQL generaba una relación en *nombre-cliente* y *ciudad-cliente*. Esto permitió declarar las variables *a* y *b* del tipo de dato correcto. Considérese ahora el problema al que se enfrenta un programador que implanta la interfaz amable con el usuario (UFI). El programador no sabe por adelantado cuáles son las proposiciones en SQL que se van a introducir. De hecho, el programador no sabe siquiera cuáles son las relaciones que van a existir cuando se ejecute el programa de UFI.

Para poder crear programas que acepten consultas en SQL arbitrarias, el Sistema R incluye un comando **describe** (describir). Este comando acepta como entrada una proposición en SQL, y su salida es:

- El número de atributos de la relación resultante.
- El tipo de dato de cada uno de los atributos de la relación resultante.

El programador de aplicaciones debe asignar en forma dinámica las variables para aceptar las tuplas de la relación resultante.

Consistencia y concurrencia

Los conceptos de colocación de candados de dos fases y de granularidad múltiple se originaron en el proyecto del Sistema R. En el Sistema R todas las transacciones mantienen por omisión todos los candados hasta que se termina de ejecutar la transacción para garantizar la serialibilidad y evitar los retrocesos en cascada. Sin embargo, también ofrece la alternativa de *grados de consistencia* que permite liberar candados antes del término de la transacción. Un programador de aplicaciones que esté diseñando una transacción para el Sistema R, puede optar por un grado más débil de consistencia, ya que al liberar pronto los candados se reduce la probabilidad de que otra transacción tenga que esperar para colocar un candado.

El protocolo de candados de granularidad múltiple ha trabajado en forma satisfactoria en el Sistema R. Sin embargo, se hizo una pequeña modificación al esquema que se presentó en el capítulo 11 para resolver el siguiente problema: a menudo sucede que una transacción acumula un gran número de candados de granularidad de registro pero no coloca un candado de granularidad de relación. A resultas de ésto, el manejador de candados tiene que mantener una tabla de candados muy grande, y el tiempo extra por mantenimiento de candados aumenta. Para enfrentarse a este tipo de situaciones, el Sistema R efectúa en forma automática una *elevación de candados*. Siempre que sea posible, los candados de granularidad de registro se intercambiarán por un candado de granularidad de relación o de archivo. Este procedimiento de elevación de candados es transparente para el programador de aplicaciones y para el usuario.

El manejador de recuperaciones del Sistema R utiliza *tanto* la doble paginación como las bitácoras. Esta estrategia combinada refleja la evolución del Sistema R desde un sistema para un solo usuario (en el que la doble paginación funciona de manera satisfactoria) hasta uno de multiusuarios. La doble pagina-

ción no se adapta bien al caso de acceso concurrente a la base de datos. El sis ma de bitácoras que se utiliza se denomina *bitácora de grabación adelant. (write-ahead logging)* y es en esencia el mismo que el de bitácora incremental con actualización inmediata que se describió en el capítulo 10.

Comentarios

El Sistema R se encuentra bien documentado en la literatura pública, debido a que es un proyecto de investigación y no un producto comercial. Aparte de su influencia sobre varios productos comerciales, el Sistema R ha dado pie a varios proyectos de investigación subsecuentes, incluyendo el proyecto de Sistema R* de base de datos distribuida y otras investigaciones que buscan extender el Sistema R para manejar objetos complejos (véase el Cap. 14).

15.1.2 SQL/Data System

El SQL/Data System (SQL/DS) es un sistema comercial de base de datos vendido por IBM; es de tamaño intermedio cuyo alcance se encuentra entre el de los sistemas de base de datos para macrocomputadoras (p. ej. DB2 e IMS) y los de base de datos para computadoras personales (p. ej., dBase III). El SQL/DS funciona con los sistemas operativos DOS/VSE o VM/SP, ambos de IBM.

El SQL/DS y el Sistema R tienen muchos puntos en común, ya que el primero se vio muy influenciado por el proyecto de investigación del Sistema R. Entre sus similaridades pueden mencionarse:

- El lenguaje de consulta SQL y el SQL incrustado.
- La precompilación.
- El control de concurrencia y manejo de transacciones.

Casi todo lo que se dijo en la sección 15.1.1 sobre estos temas se puede aplicar también al SQL/DS.

Un área importante en la que el SQL/DS va más allá del Sistema R es la capacidad para extraer información de las bases de datos IMS utilizando el lenguaje de consulta DL/1 de IMS. La función de extracción de DL/1 permite copiar datos de una base de datos IMS (jerárquica) a una parte de una base de datos SQL/DS (relacional). Una vez almacenada la información en el sistema SQL/DS, puede consultarse al utilizar el SQL. Sin embargo, no es posible modificar la base de datos IMS utilizando el SQL.

El lenguaje de definición de datos del SQL/DS incluye las proposiciones comunes del SQL: **create table** (crear tabla), **create index** (crear índice), **create view** (crear vista), etc. Además, el SQL/DS cuenta con proposiciones que permiten al administrador de la base de datos controlar la estructura física de la base de datos.

Un *espacio de base de datos* (*dbspace*, en la terminología del SQL/DS), es una sección de almacenamiento físico en disco en la que se almacenan las relaciones y sus índices. El mecanismo de seguridad del SQL/DS sirve para controlar la auto-

ridad para agregar nuevas relaciones a un espacio de base de datos. Por ejemplo, puede asignarse un espacio de base de datos personal a un usuario para almacenar sus propias relaciones. A este usuario se le asignará autorización de recursos sobre su espacio de base de datos personal.

Cuando se crea un espacio de base de datos, puede asociarse con un disco específico. El poder asignar espacios de base de datos a discos específicos en un sistema de cómputo de discos múltiples permite al administrador de base de datos distribuir los accesos a información entre varios dispositivos de disco. Esta distribución de la carga de trabajo por lo general mejora el rendimiento.

La interfaz con el usuario del SQL/DS incluye tanto un precompilador como una interfaz interactiva similar a la UFI del Sistema R. La sintaxis del precompilador es diferente de la del Sistema R, pero las construcciones y conceptos básicos del lenguaje son los mismos.

La interfaz *interactiva SQL* (ISQL) permite al usuario introducir de manera directa consultas en SQL. Además de las proposiciones de SQL, la ISQL incluye proposiciones que controlan el formato de salida de las respuestas a las consultas, comandos para grabar y editar las consultas, y proposiciones para facilitar la generación de reportes.

15.1.3 Database 2

Database 2 (DB2) es un sistema comercial de base de datos vendido por IBM para utilizarse en macrocomputadoras grandes que tienen el sistema operativo MVS.

Aunque el DB2 y el SQL/DS son distintos, existe un gran parecido entre estos dos sistemas; ambos utilizan versiones casi idénticas del lenguaje de consulta SQL, y la sintaxis del SQL incrustado es en realidad la misma. Los dos sistemas no pueden compartir datos en forma directa, pero es posible transferir datos entre ellos.

Los conceptos fundamentales del DB2 son similares a los del SQL/DS y, por tanto, similares a los del Sistema R. La importancia del DB2 radica en que cuenta con el tamaño y grado de sofisticación adecuados para manejar bases de datos muy grandes. Por ello, y a diferencia del SQL/DS, el DB2 es un competidor del IMS para el manejo de bases de datos grandes. La introducción del DB2 en 1983 subrayó la creciente importancia del modelo de datos relacional en el ambiente comercial y el decaimiento en la importancia del modelo jerárquico. Las funciones DXT del DB2 permiten transferir información del IMS al DB2 (en forma análoga a la función de extracción de DL/1 del SQL/DS).

Gran parte de lo que se dijo acerca del Sistema R y del SQL/DS puede aplicarse al DB2. A continuación se mencionan algunos aspectos interesantes del DB2 de los que no se había hablado antes.

El DB2 interactúa con tres subsistemas diferentes que funcionan con el MVS. Cada uno de éstos permite un tipo distinto de manejo de transacciones. Estos tres sistemas, IMS, TSO (la opción de tiempo compartido del MVS) y CICS (sistema de control de información a clientes, en inglés, Customer Information Control System) son anteriores al DB2. Por ello, la interacción del DB2 con estos subsistemas fue una necesidad práctica. Con todo, da pie a una aplicación inte-

resante de los conceptos de comisión de transacciones en bases de datos *distribuidas*. En el capítulo 12 se supuso que un sistema distribuido consiste de computadoras conectadas por medio de una red. Dentro del DB2, los subsistemas coexisten dentro de la misma máquina. Sin embargo, como en el caso de los sistemas distribuidos, es preciso que los subsistemas estén de acuerdo, ya sea para cometer o para abortar las transacciones que utilicen los servicios de más de un subsistema. Esto se logra dentro del DB2 al utilizar el protocolo de comisión de dos fases que se vio en el capítulo 12.

La *facilidad de manejo de consultas* (QMF, *query management facility*) del DB2 es una interfaz que permite al DB2 interactuar con la base de datos al emplear:

- SQL.
- QBE.
- Un generador de informes.

El lenguaje QBE se describió en el capítulo 3. En un principio, el QBE era un sistema de base de datos completo desarrollado en el IBM T. J. Watson Research Center. La característica más notable del sistema QBE es su lenguaje de consultas. Dado que tanto el QBE como el SQL son lenguajes de consulta relacionales, es factible que un sistema de base de datos ofrezca ambos a sus usuarios.

Al utilizar la QMF, los usuarios de DB2 pueden compartir información aun en el caso en que uno utilice SQL y otro QBE. Los usuarios de ambos lenguajes pueden disponer también del generador de informes. La QMF genera en forma automática un formato para el reporte, y el usuario puede alterar este formato si lo desea. En la actualidad, también es posible que el SQL/DS aproveche la función QMF.

15.1.4 Oracle

El sistema de base de datos Oracle, vendido por la Oracle Corporation, es un sistema que aunque no fue desarrollado por la IBM, se apega fielmente al lenguaje de consultas y a la interfaz con el usuario del producto SQL/DS de IBM.

Oracle funciona en muchos sistemas diferentes, incluyendo las computadoras DEC VAX y las similares a la IBM-370. Como los sistemas IBM antes descritos, Oracle utiliza el lenguaje de consulta SQL y permite que los programas de aplicaciones utilicen el SQL incrustado. Se ofrece SQL incrustado para los lenguajes Cobol, Fortran y C. Además de una técnica de precompilador para manejar el SQL incrustrado, Oracle incluye una *interfaz de llamadas* que interpreta las proposiciones de SQL en el momento de la ejecución, en vez de compilarlas. Las llamadas de la interfaz de llamadas corresponden directamente a las proposiciones del precompilador. Aunque la interfaz de llamadas elimina el paso de precompilación cuando se prepra un programa de aplicaciones, es preferible utilizar la técnica de precompilación para los programas que se van a ejecutar varias veces. Como casi todos los sistemas comerciales, Oracle incluye un generador de informes. Sin embargo, las características de este generador son muy diferentes

de las de los sistemas que ya se mencionaron (excepto por el hecho de utilizar todos el SQL).

Oracle utiliza índices de árbol B^+ con una opción de *compresión de llaves*. La compresión de llaves es una técnica para ahorrar espacio. En vez de guardar la llave completa en el índice, sólo se guarda una parte que baste para identificar el apuntador que es preciso seguir. Esta técnica ahorra una cantidad considerable de espacio cuando las llaves son largas (como en el caso de nombres de personas), pero tiene la desventaja de que para determinar el valor completo de una de ellas es preciso leer un registro de la base de datos.

La recuperación de caídas del Oracle se basa en *archivos de imagen previa*. El archivo de imagen previa contiene copias de los bloques de la base de datos que hayan sido modificados por las transacciones sin cometer. Estas copias se usan para el retroceso de las transacciones abortadas. La técnica es similar a la de doble paginación que se presentó en el capítulo 10.

El control de concurrencia de Oracle utiliza candados y permite un grado todavía mayor de concurrencia por medio del uso del archivo de imagen previa. Esta técnica es una forma de control de concurrencia de multiversión en el que sólo se permiten dos versiones. Supóngase que la transacción T_1 solicita acceso para lectura a los datos que T_2 está modificando. Con los protocolos para candados normales, T_1 tendría que esperar a que T_2 libere el candado. Oracle, por el contrario, permite a T_1 leer los valores previos a la modificación que realiza T_2. Estos valores están almacenados en el archivo de imagen previa. Si no se desean los datos de la imagen previa, T_1 puede solicitar un candado, lo que la obligará a esperar a que cualquier operación de escritura que se esté efectuando en forma concurrente libere la información.

15.1.5 Ingres

En la misma época en que el IBM San José Research Laboratory estaba desarrollando el prototipo del Sistema R, un grupo de investigadores en la University of California at Berkeley desarrollaba un sistema experimental de base de datos llamado Ingres. Aunque ambos proyectos partieron de los conceptos del modelo de datos relacional, presentan diferencias sustanciales en cuanto al diseño del sistema y su interfaz con el usuario.

El proyecto de investigación Ingres condujo al desarrollo de un producto comercial del mismo nombre, vendido por Relational Technology, Inc. En esta sección se analizará la versión académica de Ingres, aunque muchos de los comentarios pueden aplicarse también a la versión comercial.

Estructura general del sistema

Ingres se desarrolló en una computadora PDP-11 con el sistema operativo UNIX. Este sistema operativo tuvo una influencia considerable sobre la estructura interna de Ingres. Se utilizan varios procesos, cada uno de los cuales se encarga de una tarea específica:

- Formulación de consultas, interacción con el usuario.

- Análisis lexicológico y sintáctico, optimización de consultas.
- Ejecución de consultas.
- Mantenimiento de índices.

La solicitud que hace el usuario se turna de proceso a proceso según sea necesario.

Este diseño de multiprocesos permite la concurrencia dentro de Ingres mismo, ya que es posible, por ejemplo, efectuar el análisis sintáctico de una consulta mientras se ejecuta otra. Sin embargo, la comunicación interproceso (por medio de los conductos Unix) conlleva un cierto tiempo extra. Históricamente, uno de los factores que más influenciaron el diseño de Ingres fue la limitación que tenía la PDP-11 de espacios de direccionamiento de 64 kilobytes para cada proceso individual.

Ingres introdujo el lenguaje de consulta Quel y una incrustación de éste dentro del lenguaje huésped C llamada Equel. Ya se habló del Quel en el capítulo 3. Gran parte de la sintaxis de Equel tiene una forma similar a la de SQL incrustado. Se utilizan los caracteres "##" para distinguir a una proposición de Quel que aparece dentro de un programa en C. Los caracteres "##" se emplean también para marcar las declaraciones en C que se refieran a variables a las que se hace referencia dentro de una proposición en Quel.

Equel difiere de SQL incrustado en su equivalente de los cursores de SQL. Recuérdese que en el SQL incrustado, se *abre* un cursor para una consulta y a continuación se obtienen tuplas individuales por medio de una llamada de búsqueda (**fetch**). El Equel utiliza un enfoque más estructurado en el que se ejecuta una sección específica del código, una vez por cada tupla del resultado de una consulta en Quel. La figura 15.1 muestra un ejemplo de programa en Equel c imprime el nombre y la ciudad de los clientes que tienen más de x dólares .

```
## char a [20], b [20];
## float x;
main () {
    .
    .
    .
## range of d is depósito
## range of c is cliente
## retrieve a = c.nombre-cliente, b = c.ciudad-cliente
## where d.saldo > x
    # { printf ("%5, %7.2f\n", a, b)
    # }
    .
    .
    .
```

Figura 15.1 Ejemplo de programa en Equel.

una cuenta. El código delimitado por "#{" y "#}" se ejecuta una vez por cada tupla del resultado. Los valores de la tupla para cada uno de los atributos de la relación resultante se almacenan en las variables de la cláusula **retrieve** (obtener) (*a* y *b* en la figura 15.1).

Ingres no permite la precompilación de las consultas en Quel, aun en el caso en que la consulta está incrustada en un programa en C (Equel). La optimización de consultas se realiza como parte de la interpretación de éstas, en vez de hacerse durante la precompilación. Una técnica de optimización de consultas introducida por Ingres, y que ha resultado muy útil, se denomina *descomposición*. Se construye una *gráfica de consultas* cuyos nodos son relaciones y cuyas aristas representan productos theta o naturales. Si una arista representa el producto de una relación pequeña con otra, el resultado esperado es otra relación pequeña. Por tanto, los productos de este tipo se procesan primero. Cuando no queden más de estos productos, se lleva a cabo la descomposición.

Para ilustrar la descomposición se utilizará la parte de una gráfica de consultas que se muestra en la figura 15.2. Esta gráfica ilustra el producto de tres relaciones, r_1, r_2, y r_3 . Los esquemas correspondientes son R_1, R_2 y R_3. La relación r_2 se descompone de la siguiente manera:

- Se construye un índice por r_1 para $R_1 \cap R_2$ (a menos que ya exista ese índice).
- Se construye un índice por r_3 para $R_3 \cap R_2$ (a menos que ya exista ese índice).
- Se descompone r_2 en varias relaciones. Cada relación tiene el esquema R_2 y contiene sólo una tupla de r_2. Se produce una gráfica de consulta para cada una de las relaciones descompuestas. En cada una de estas gráficas una relación descompuesta toma el lugar de r_2.
- Se procesa cada una de las gráficas de consulta y se calcula la unión de los resultados.

Puesto que todas las gráficas de consulta que se generan al aplicar la descomposición tienen la misma estructura, es fácil procesarlas.

15.2 Sistemas de red

El modelo de red es la base de la mayor parte de los sistemas antiguos de base de datos, y de algunos sistemas recientes. En esta sección se analizarán brevemente dos sistemas de red populares, Total e IDMS.

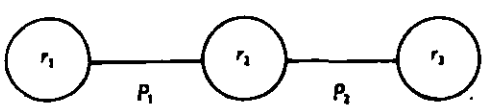


Figura 15.2 Parte de una gráfica de consultas de Ingres.

15.2.1 Total

El sistema de base de datos Total data de finales de la década de 1960. Funciona en una gran variedad de máquinas que van desde minicomputadoras hasta macrocomputadoras.

Aunque Total se basa en el modelo DBTG, su lenguaje de consulta difiere del que se presentó en el capítulo 4. El lenguaje presentado en el capítulo 4 está basado en el estándar DBTG. El lenguaje de Total, aunque tiene una función similar, utiliza una sintaxis diferente.

Todas las proposiciones del lenguaje de manipulación de datos Total deben incrustarse en un lenguaje huésped (Cobol, PL/1, Fortran o RPG). Se hace una llamada al procedimiento *database* con parámetros que especifican la proposición en lenguaje de manipulación de datos. Estos comandos son similares a los comandos **find** (encontrar), **find owner** (encontrar dueño), y **find next** (encontrar el siguiente) que se vieron en el capítulo 4, aunque la terminología que utiliza Total tiene su origen en algunos de los conceptos del lenguaje DL/1 del IMS. Los tipos dueños se denominan conjuntos de datos de entrada única y los tipos miembros se llaman conjuntos de datos de entrada variable.

La implantación física interna de Total y los esquemas de acceso que se utilizan no han sido publicados.

15.2.2 IDMS

IDMS es un sistema de base de datos de red desarrollado por Cullinane Database Systems, Inc. IDMS sigue de cerca el modelo DBTG tal y como se presentó en el capítulo 4; incluye un lenguaje de manipulación de datos detallado que permite al diseñador de base de datos un grado de control considerable sobre la ganización física de la base de datos. El lenguaje de manipulación de datos tie. características idénticas (o casi idénticas) al lenguaje de manipulación de datos DBTG (DBTG DML) que se describió en el capítulo 4. Se han incluido comandos adicionales para facilitar la labor del programador y para que los programadores experimentados sean capaces de redactar consultas más eficientes.

Un ejemplo de las características de IDMS que no se mencionaron en el capítulo 4 es el comando **obtain** (obtener) que combina en una sola solicitud los comandos **find** y **get.** Puede agregarse una cláusula **where** opcional al comando **obtain** para localizar y obtener el siguiente registro que satisfaga el predicado de la cláusula **where.** Esto le ahorra al programador la tarea de escribir una prueba explícita para un registro que se haya localizado por medio del comando **find.**

Las notas bibliográficas hacen referencia a algunos documentos que describen en forma detallada el manejo interno de archivos y las técnicas de indización empleadas por el IDMS.

15.3 Sistemas jerárquicos

El modelo jerárquico tiene relevancia en primer término debido a la importancia del sistema de base de datos IMS de IBM. En esta sección se hablará tanto del IMS como de otro sistema jerárquico muy utilizado, el Sistema 2 000.

15.3.1 IMS

El sistema de Manejo de Información (IMS, Information Management System) de IBM es uno de los sistemas de base de datos más antiguos y más utilizados. Trabaja con el sistema operativo MVS en máquinas grandes IBM-370 y similares. Por tradición, las bases de datos más grandes han sido bases de datos IMS, por lo que las personas encargadas del desarrollo de este sistema fueron de las primeras en tener que enfrentarse a los problemas de concurrencia, recuperación, integridad y procesamiento eficiente de las consultas. Al pasar por varias versiones, IMS fue adquiriendo un gran número de funciones y opciones y, por esta razón, es un sistema de gran complejidad; aquí se analizarán sólo algunas de sus características.

IMS está basado en el modelo de datos jerárquico (véase el Cap. 5). Las consultas a las bases de datos IMS se hacen por medio de llamadas incrustadas en un lenguaje huésped. Las llamadas incrustadas son parte del lenguaje de base de datos DL/1 del IMS. El lenguaje que se utilizó en el capítulo 5 es una forma simplificada de DL/1.

Puesto que el rendimiento es de primordial importancia en las bases de grandes datos, IMS permite al diseñador de base de datos un gran número de opciones dentro del lenguaje de definición de datos. el diseñador define al esquema de la base de datos como una jerarquía física. Pueden definirse varios subesquemas (o vistas) construyendo una jerarquía lógica a partir de los tipos de registro que constituyen el esquema. El lenguaje de definición de datos incluye una gran variedad de opciones (tamaños de bloques, campos apuntadores especiales, etc.) que permiten al administrador de la base de datos "afinar" al sistema con el fin de mejorar su rendimiento.

IMS cuenta con varios esquemas de acceso a registros:

- HSAM (método jerárquico de acceso secuencial, *hierarchical sequential-access method*), que se utiliza para archivos cuya organización física es secuencial (como los archivos en cinta). Los registros se almacenan físicamente en preorden.
- HISAM (método jerárquico de acceso secuencial indizado, *hierarchical indexed-sequential access method*), que es una organización secuencial indizada en el nivel de raíz de la jerarquía.
- HIDAM (método jerárquico indizado de acceso directo, *hierarchical indexed direct-access method*), que es una organización indizada en el nivel de la raíz con apuntadores a los registros hijos.
- HDAM (método jerárquico de acceso directo, *hierarchical direct-access method*), que es similar al HIDAM pero con acceso por cálculo de dirección al nivel de la raíz.

La versión original del IMS antecede al desarrollo de la teoría de control de concurrencia. Las primeras versiones del IMS contaban con una forma sencilla de control de concurrencia: sólo podía ejecutarse un programa que incluyera ac-

tualizaciones a la vez. Sin embargo, podían ejecutarse de manera concurrer cualquier cantidad de aplicaciones de lectura y una aplicación de actualizaci Esto permitía a las aplicaciones leer valores actualizados no cometidos y ejecutarse en forma no serializable. La única opción disponible para las aplicaciones que requerían mayor protección contra las anomalías del procesamiento concurrente era el acceso exclusivo a la base de datos.

Versiones posteriores del IMS incluyeron una *función de aislamiento de programas* más complejos, que permitía un mejor control de la concurrencia y técnicas de recuperación de transacciones, a su vez más complejas (p. ej., las bitácoras). Estas características se hicieron cada vez más importantes conforme se popularizó el uso de transacciones en línea en vez de las transacciones por lote que predominaban en un principio.

La necesidad de un procesamiento de transacciones de alto rendimiento condujo a la introducción del *IMS Fast Path* (ruta rápida). El Fast Path utiliza una alternativa de organización física diseñada de tal manera que permite que las partes más activas de la base de datos residan en la memoria principal. En vez de forzar la salida de actualizaciones al disco después de una transacción (como lo hace el IMS estándar), la actualización se posterga hasta el siguiente punto de verificación o de sincronización. En el caso de una caída, el subsistema de recuperación debe repetir todas las transacciones cometidas cuyas actualizaciones no se grabaron en disco. Estos y otros "trucos" permiten una velocidad de procesamiento de transacciones muy alta.

Una base de datos IMS puede consultarse, pero no actualizarse, utilizando el SQL. La función DXT del DB2 y la función de extracción de DL/1 del SQL/DS permiten almacenar datos de IMS en una base de datos relacional en la que pueden realizarse consultas en SQL.

Los detalles completos del IMS están más allá del alcance de este breve panorama. Aunque el IMS es un sistema antiguo y la importancia del modelo jerárquico disminuye, la historia y la evolución del IMS constituyen un marco de referencia interesante para el estudio del desarrollo de los conceptos de sistemas de base de datos.

15.3.2 Sistema 2 000

El Sistema 2 000 es un sistema jerárquico de base de datos desarrollado en un principio por la MRI Corporation y distribuido en la actualidad por Intel. El Sistema 2 000 puede utilizarse en computadoras tipo IBM-370, Univac 1 100 y en CDC 6 000 y Cyber. Aunque utiliza el mismo modelo de datos que el IMS, las características del lenguaje que ofrece el Sistema 2 000 presentan varias diferencias interesantes con respecto al lenguaje DL/1 el IMS.

El concepto fundamental del lenguaje de definición de datos DL/1 es la relación padre hijo uno a muchos. En el ejemplo bancario, supóngase que existe una relación uno a muchos entre *cliente* y *cuenta*. En el diseño de la base de datos, *cuenta* se definiría como hijo de *cliente*. Otra forma de considerar esta relación uno a muchos es, como un registro que tiene muchos campos repetidos. Dentro del registro *cliente* de un cliente determinado puede almacenarse el conjunto de cuen-

tas que pertenecen a ese cliente si se crea un campo *cuenta* repetido. Este enfoque de campos repetidos es el fundamento del lenguaje de definición de datos del Sistema 2 000.

En el ejemplo bancario del capítulo 5, el registro *cliente* tiene los campos *nombre-cliente, calle* y *ciudad-cliente,* y el registro *cuenta* tiene los campos *número-cuenta* y *saldo.* A continuación se muestran las proposiciones que corresponden a esta parte del sistema de base de datos utilizando el lenguaje de definición de datos del Sistema 2 000:

1* *nombre-cliente* (name);
2* *calle* (name);
3* *ciudad-cliente* (name);
4* *cuenta* (repeating group);
5* *número-cuenta* (integer);
6* *saldo* (money);

Los tipos de datos *name* (nombre), *integer* (entero) y *money* (dinero), entre otros, vienen incluidos en el Sistema 2 000. Los campos se especifican como *key* (llave) o *nonkey* (no llave). El Sistema 2 000 construirá un índice por cada uno de los campos llave.

Considérese la consulta "encontrar los nombres de todos los clientes que tienen una cuenta cuyo saldo es superior a $10 000, y las ciudades donde viven". Si se utiliza el lenguaje similar al DL/1 del capítulo 5, es preciso revisar todos los registros de cuentas de cada cliente, y para ello se escribe un ciclo **while** en el lenguaje huésped. El lenguaje de consultas del Sistema 2 000, QUEST, permite escribir la consulta utilizando una sola proposición.

list *nombre-cliente, ciudad-cliente*
while *cliente* **has** *saldo* > 10 000

También es posible incrustar las consultas del Sistema 2 000 en un lenguaje huésped. Un precompilador procesa el programa. El análisis sintáctico de las consultas se hace durante la fase de precompilación. Al igual que la mayor parte de los sistemas comerciales, el Sistema 2 000 incluye funciones que auxilian en la generación de reportes.

15.4 Sistemas de bases de datos para microcomputadoras

La característica fundamental de los sistemas de base de datos para microcomputadoras, es su sencillez. La capacidad limitada de las computadoras personales restringe tanto el tamaño de la base de datos como el grado de complejidad del sistema. Aunque casi todos los sistemas de base de datos se diseñan pensando en la facilidad de uso, la importancia de este factor en el mercado de las computadoras personales es extraordinaria, ya que los usuarios no pueden contar con

la ayuda de un administrador de base de datos experimentado. Cada uno d usuarios de un sistema de base de datos de microcomputadora funge como ... ministrador de base de datos.

A continuación se compararán las características comunes de los sistemas de base de datos para microcomputadoras con las de los sistemas más grandes:

- **Modelo de datos.** Dado que los sistemas de base de datos para microcomputadoras son relativamente nuevos, casi todos están basados en el modelo relacional. Algunos de esos sistemas deben considerarse más bien *tabulares*, ya que, aunque utilizan tablas, son demasiado primitivos para llamarse relacionales.
- **Lenguaje de consultas.** Es posible que aun los lenguajes de más alto nivel que se analizaron en este texto sean demasiado complejos para el usuario casual de un sistema de base de datos para microcomputadora. Muchos de los lenguajes se basan en una interfaz de forma, con la cual el usuario puede interactuar con el sistema llenando una forma.
- **Implantación física.** Para los implantadores de un sistema, uno de los factores importantes es el espacio que ocupa el código objeto de un sistema de base de datos para microcomputadora. Si se reduce el espacio requerido, es posible utilizar el sistema en máquinas que cuenten con menos memoria principal. Esto puede tener una influencia determinante sobre el mercado potencial. Por esta razón son pocos los sistemas que utilizan técnicas de manejo de memoria y de indización complejas. Por lo general se elige un solo tipo de índice, y la optimización de consultas, cuando lleva a cabo, es rudimentaria.
- **Recuperación.** Muchos sistemas no cuentan con subsistema de recuperación. El usuario tiene la responsabilidad de sacar copias de respaldo de sus datos en forma regular.
- **Concurrencia.** No se requiere control de concurrencia para computadoras personales de un solo usuario.

La distinción entre los sistemas de base de datos para microcomputadoras y los sistemas más grandes se hace menos clara. Los primeros sistemas de base de datos para microcomputadoras no eran mucho más que interfaces para lograr acceso a un solo archivo de registros de longitud fija. Al crecer la capacidad de las computadoras personales, ha crecido también la complejidad de los sistemas de base de datos para microcomputadora. De hecho, han comenzado a aparecer versiones de algunos de los sistemas de base de datos de gran escala en las computadoras personales de mayor tamaño. Un ejemplo de este tipo de sistemas es el SQL/RT de la computadora personal IBM RT.

En esta sección se examinará el dBase-III, que ha tenido gran éxito en el mercado de los sistemas de base de datos para microcomputadora, y el Lotus 1-2-3, que es un sistema de "base de datos de hoja electrónica".

15.4.1 dBase-III

El dBase-III de Ashton-Tate es una actualización del exitoso producto comercial dBase-III. Incluye tanto un lenguaje de manipulación de datos como uno de programación de aplicación general. Aunque las características de estos lenguajes están basadas en el Cobol, Pascal y los de base de datos que se estudiaron aquí, el lenguaje del dBase-III es único. El lenguaje de programación es necesario para todas las consultas, con excepción de las más sencillas, debido a que el lenguaje de manipulación de datos, por sí solo, es menos poderoso que el álgebra relacional.

La selección y la proyección pueden expresarse en dBase-III al emplear el comando **display** (exhibir). Considérese por ejemplo la consulta: "encontrar los nombres de todas las sucursales en las que Jones tiene una cuenta". En primer término, se escribe la proposición:

use *depósito*

para indicar que se desea utilizar el archivo *depósito* que contiene la tabla *depósito*. La proposición:

display all off for *cliente* = "Jones", *nombre-sucursal*

muestra la respuesta a la consulta. La palabra clave **all** (todos) hace que se exhiban todos los registros que satisfacen a la consulta, no sólo uno de ellos. La palabra clave **off** suprime la impresión de números de registro.

La proposición **delete** (borrar) utiliza una sintaxis similar. Una característica interesante del borrado en dBase-III es que a los registros borrados sólo se les *marca* como tales, pero en realidad no se eliminan de la base de datos. Esto permite a un usuario nulificar un borrado erróneo por medio de la proposición **recall** (recordar). Los registros borrados se eliminan de la base de datos sólo cuando el usuario hace que se ejecute la proposición **pack** (empacar).

La proposición **append** (agregar) hace que el dBase-III entre al modo de inserción. En la pantalla se exhibe una forma con un espacio en blanco por cada campo de un registro de la tabla que se especifica en la proposición **use** (utilizar) más reciente. El usuario llena la forma.

El cálculo de productos en dBase es más laborioso que en los lenguajes de consulta que se describieron en el capítulo 3. El usuario debe crear una tabla nueva que contenga el producto. Se requiere un comando **select** (elegir) para poder especificar varias tablas en las proposiciones **use**. Cabe hacer notar que esta forma de utilizar la palabra **select** no corresponde al significado que tiene en SQL ni al de la selección en el álgebra relacional. La secuencia de proposiciones que se requiere para calcular:

$$depósito \times cliente$$

es:

select 2

```
use depósito
select 1
use cliente
join with depósito to cd for nombre-cliente = cliente — > nombre-cliente;
     fields nombre-sucursal, número-cuenta,
     cliente- > nombre-cliente, saldo, calle, ciudad-cliente
use cd
```

Las dos proposiciones **select** y **use** especifican las tablas a las que se va a tener acceso. Se realiza el producto de la tabla *cliente* con *depósito* en la proposición **join**, y el tipo de producto que se lleva a cabo es theta, seguido de una proyección. La cláusula **for** (para) de esta proposición especifica el predicado de producto. La cláusula **fields** (campos) especifica los atributos en los que se va a proyectar el producto theta. Nótese que el dBase-III utiliza la notación:

nombre-tabla — > nombre-atributo

en vez de la notación

nombre-relación.nombre-atributo

que se utilizó en los capítulos 3 y 6. El resultado del producto es una tabla nueva, *cd*, que contiene al producto, puede usarse la proposición *display* para exhibir el resultado.

Si son necesarios varios productos para responder a una consulta, y se sigue un método similar al antes descrito, será preciso crear un gran número de archivos temporales. A diferencia de los sistemas grandes de base de datos en los que el optimizador de consultas se encarga de calcular los productos de manera eficiente, el dBase obliga al usuario a escribir un programa para calcular en form eficiente el producto de varias relaciones.

El lenguaje de programación del dBase incluye el acceso a la base de datos como parte integral del lenguaje, en vez de incrustarlo en un lenguaje ya existente por medio de caracteres de escape especiales (como el "$" del Sistema R o el "#" del Ingres). No se presentará aquí el lenguaje de programación de dBase-III completo. Tiene las estructuras de control usuales (**if-then-else, while, case**). Además, tiene un comando **find** (encontrar) que permite localizar los registros que tienen un valor determinado en el campo que se especifique. Sin embargo, este comando funciona sólo para los campos indizados. Para los demás campos es preciso emplear el comando **locate** (localizar) que realiza una búsqueda lineal. Una vez localizado el registro con cualquiera de los dos comandos, el programa puede realizar operaciones con los valores de los campos del registro.

Debido a la capacidad limitada del lenguaje de consulta del dBase, es necesario utilizar el lenguaje de programación del dBase con más frecuencia que el SQL incrustado del Sistema R. El programador de dBase debe estar consciente de la eficiencia de la estrategia de procesamiento de consultas que va a utilizar, mientras que el programador de SQL incrustado puede dejar la mayor parte de las consideraciones de eficiencia al optimizador de consultas.

	A	B	C	D	E
1	millas recorridas	precio (dlls/gal)	importe	galones	mi/gal
2				C2/B2	A2/D2
3				C3/B3	A3/D3
4				C4/B4	A4/D4
5				C5/B5	A5/D5

Figura 15.3 Hoja electrónica de rendimiento de gasolina-fórmulas.

15.4.2 Sistemas de base de datos de hoja electrónica

El Visicalc, producido por Software Arts, Inc., fue el primer lenguaje de hoja electrónica exitoso. A diferencia de los lenguajes de programación tradicionales, la naturaleza del lenguaje de hoja electrónica es bidimensional. Una hoja electrónica es un arreglo bidimensional de celdas. Para programar con una hoja electrónica es necesario definir relaciones matemáticas entre las celdas. El sistema de hoja electrónica mantiene estas relaciones al irse introduciendo los datos. Por tanto, a cada celda puede estar asociada una *fórmula* o un *valor*.

La figura 15.3 muestra las fórmulas de una hoja electrónica sencilla que mantiene los registros de rendimiento de gasolina de un automóvil. En una hoja de cálculo, las columnas reciben los nombres A, B, ..., y las hileras 1, 2, ...,. En la hoja aparecen las fórmulas para calcular el número de galones adquiridos y las millas recorridas por galón. Obsérvese, por ejemplo, la celda D2, que contiene la fórmula C2/B2. Esto quiere decir que el valor de D2 se calcula dividiendo el valor que está en la celda C2 entre el que está en B2. Supóngase que se introducen en la hoja los valores que se muestran en la figura 15.4. La hoja electrónica que ve el usuario se muestra en la figura 15.5. En vez de las fórmulas, el usuario ve el resultado de las mismas. Existen comandos especiales para exhibir y modificar las fórmulas.

El objetivo de esta sección no es describir en la forma detallada la programación de hojas electrónicas. Lo que se quiere hacer resaltar es que el conjunto rectangular de celdas de una hoja electrónica puede considerarse como una tabla. Esto sugiere la combinación de los lenguajes de hoja electrónica y de base de datos dentro de una sola estructura. Este tipo de lenguaje se ilustrará por medio del producto Lotus 1-2-3 de la Lotus Development Corporation.

El esquema de una tabla 1-2-3 se define al introducir el nombre de campo (atributo) para cada una de las columnas de la hilera 1. La hilera 1 es la única

	A	B	C	D	E
1	millas recorridas	precio (dlls/gal)	importe	galones	mi/gal
2	351	1.099	9.00		
3	292	1.119	8.25		
4	302	1.099	8.70		
5	289	1.079	9.10		

Figura 15.4 Hoja electrónica de rendimiento de gasolina: introducción de valores.

	A	B	C	D	E
1	millas recorridas	precio (dlls/gal)	importe	galones	mi/gal
2	351	1.099	9.00	8.19	42.0
3	292	1.119	8.25	7.37	39.6
4	302	1.099	8.70	7.92	38.0
5	289	1.079	9.10	8.43	34.3

Figura 15.5 Hoja electrónica de rendimiento de gasolina: resultado que ve el usuario.

que puede utilizarse para esto. No es preciso que el usuario se preocupe mucho por los tipos de datos. Todo lo que tiene que hacer es decidir el número de caracteres que va a necesitar para desplegar los valores en cada columna. Así, de hecho, todos los dominios son del tipo *cadena*, y su longitud la especifica el usuario. Los datos se introducen en la base de datos de hoja electrónica de la misma manera que se hace con los datos de una hoja de cálculo ordinaria.

Las tablas del Lotus 1-2-3 difieren de las hojas de cálculo sencillas en cuanto a que pueden efectuarse en ellas clasificaciones y búsquedas. El comando de clasificación (/DS) requiere como entradas la especificación de un rango continuo para clasificar y la llave de clasificación (las columnas según las cuales se van a clasificar las hileras).

Para obtener información de una tabla de Lotus 1-2-3 se utiliza el comando/DQ (data query, consulta de datos). Como en el caso del comando de clasificación, es preciso especificar un intervalo contiguo de hileras para la búsqueda. El predicado se especifica creando una plantilla en forma de tabla, donde los encabezados de las columnas son los nombres de los campos que deben tener un valor específico en el predicado. La forma de introducir el predicado a esta plantilla recuerda al QBE (consulta por ejemplo). La hilera número i de la plantilla especifica un predicado P_i. La consulta es:

$$P_1 \wedge P_2 \wedge \cdots$$

Cada P_i es una conjunción de condiciones referentes a campos individuales. Si aparece una constante C en la columna i de una hilera, sólo se obtendrán las hileras que contengan el valor C en la columna i. La figura 15.6 muestra la consulta, "encontrar todos los registros en los que el precio sea $1.099".

	A	B	C	D	E
1	millas recorridas	precio (dlls/gal)	importe	galones	mi/gal
2	351	1.099	9.00	8.19	42.0
3	292	1.119	8.25	7.37	39.6
4	302	1.099	8.70	7.92	38.0
5	289	1.079	9.10	8.43	34.3
6		precio (dlls/gal)			
		1.099			

Figura 15.6 Consulta para localizar las hileras en las que el precio sea 1.099.

	A	B	C	D	E
1	millas recorridas	precio	importe	galones	mi/gal
2	351	1.099	9.00	8.19	42.0
3	292	1.119	8.25	7.37	39.6
4	302	1.099	8.70	7.92	38.0
5	289	1.079	9.10	8.43	34.3
6	millas recorridas	precio	importe	galones	mi/gal
					+ E2 > 40 *y* E2 < 50

Figura 15.7 Consulta para localizar las hileras en las que el rendimiento esté entre 40 y 50.

Para especificar condiciones que involucren comparaciones distintas de la igualdad pueden utilizarse los símbolos ">", "<", etc. No hay un equivalente directo de las variables de dominio del QBE, pero la colocación en la hoja de cálculo del valor de un campo en la primera hilera puede utilizarse como una forma de variable de dominio. La figura 15.7 muestra la consulta, "encontrar todos los registros en los que el rendimiento fue mayor que 40 y menor que 50 millas por galón".

En todas las consultas de 1-2-3, las hileras que forman el resultado se copian en un *rango* de salida, es decir, una sección de la hoja de cálculo que el usuario especifica con anterioridad como parámetro de un comando/DQ.

Sólo se mencionaron aquí algunas de las características del comando/DQ. No obstante, es necesario recalcar que aunque no es difícil expresar la mayor parte de las consultas básicas, el lenguaje de consulta no tiene el poder ni la amplitud de los lenguajes de consulta que se describieron en los capítulos 3, 4 y 5.

Notas bibliográficas

La distinción entre los sistemas de base de datos relacionales y tabulares se analiza en [Codd 1982].

En un gran número de trabajos publicados se presentan diversos aspectos del Sistema R. Astrahan et al. [1976], Astrahan et al. [1979], y Blasgen et al. [1979] presentan un panorama general del Sistema R. Chamberlin et al. [1976] introdujo el lenguaje SQL. Chamberlin [1980], Reisner [1977] y Reisner et al. [1975] presentan un análisis de los factores humanos del SQL. Eswaren et al. [1976], Gray [1978] y Gray et al. [1975, 1976] comentan el control de concurrencia en el Sistema R. Gray et al. [1981a] analiza la técnica de recuperación de caídas del Sistema R. Griffiths y Wade [1976], Fagin [1978], y Chamberlin et al. [1978] analizan la seguridad y las autorizaciones en el Sistema R. Lorie y Wade [1979] y Selinger et al. [1979] explican como se lleva a cabo la compilación de consultas en el Sistema R. Chamberlin et al. [1981] presentan un panorama del Sistema R que se escribió después del término del proyecto.

Williams et al. [1982] presenta un panorama general de R*, que es una versión distribuida experimental del Sistema R. Lindsay et al. [1980] presentan un panorama más detallado de la forma como el Sistema R* enfoca el manejo distribuido de datos. Traiger et al. [1982] analizan la recuperación; Lindsay [1981] ha-

15

Estudios de casos

En este capítulo se analizarán algunos sistemas de base de datos selectos que, o bien se distribuyen en forma comercial, o son sistemas experimentales que han tenido un impacto importante. No se proporcionarán los detalles completos de los sistemas elegidos. Más bien, se harán resaltar las características más importantes de los sistemas.

Dado el número de sistemas que pueden adquirirse en el mercado, y la gran cantidad de sistemas que se liberaron mientras se escribía este libro, la descripción que sigue dista mucho de ser completa. Los sistemas se eligieron para ilustrar los conceptos presentados en este texto. En consecuencia, el hecho de haber elegido un sistema no implica que se recomiende el producto.

15.1 Sistemas relacionales

El término *relacional* se ha aplicado a un gran número de sistemas, incluso a algunos que, a pesar de utilizar *tablas*, no siguen el espíritu de la definición original del modelo de datos relacional. Para que un sistema pueda considerarse relacional, es necesario que cumpla con las siguientes condiciones:

- Los datos se almacenan en tablas.
- El usuario no puede distinguir apuntadores o ligas.
- El lenguaje de consulta es completo en cuanto a las relaciones.
- Las consultas pueden expresarse sin utilizar iteraciones o proposiciones recursivas.

Los sistemas cuyas estructuras de datos básicas son tablas, pero que imponen restricciones en cuanto a las uniones que se permiten, no se consideran relacionales. Para este tipo de sistemas se utilizará el término *tabulares*.

15.1.1 Sistema R

El proyecto de investigación del sistema R se inició en el IBM San José Research Laboratory en 1974. El objetivo del proyecto era demostrar la aplicación práctica del modelo de datos relacional, que en ese entonces se acababa de proponer. Para ello, era necesario demostrar que el modelo relacional era apropiado para una interfaz con el usuario, y descubrir formas de implantar un sistema relacional que procesara de manera eficiente las consultas.

El éxito de estos trabajos ha quedado demostrado tanto por los productos comerciales que se derivaron de las investigaciones sobre el Sistema R como por el hecho de que dichas investigaciones establecieron un buen número de conceptos fundamentales. El proyecto Sistema R se terminó en 1979. Poco tiempo después IBM anunció el producto de base de datos *SQL/Data System* (véase la sección 15.1.2) y formó el proyecto de investigación de la base de datos distribuida R* para extender las investigaciones del Sistema R a las bases de datos distribuidas.

Entre las contribuciones más importantes del proyecto Sistema R están:

- El lenguaje de consulta SQL.
- La compilación y optimización de consultas.
- La integración de un lenguaje relacional y un lenguaje de programación convencional.
- La serialibilidad y el protocolo de candados de dos fases.
- El protocolo de granularidad múltiple para candados.

El SQL se describió en forma detallada en el capítulo 3, y los candados de granularidad múltiple se presentaron en el capítulo 11.

Estructura general del sistema

La arquitectura interna del Sistema R consta de dos componentes principales:

- Sistema de almacenamiento relacional (RSS, *Relational Storage System*), que se encarga de manejar la memoria (almacenamiento en disco y memoria principal), de la recuperación de caídas, y del control de concurrencia.
- Sistema de datos relacional (RDS, *Relational Data System*), que se encarga de vistas, autorizaciones e integridad.

El RSS permite tener acceso registro por registro a las relaciones del modelo lógico. El acceso a niveles más altos se logra por medio de la interfaz RDS. Ésta acepta expresiones en SQL.

Sobre la base del RDS puede construirse un número arbitrario de interfaces con la base de datos compartida del Sistema R. La que se utiliza con más frecuencia es la interfaz amable con el usuario (UFI, en inglés, User-Friendly Interface). La UFI es un programa que acepta proposiciones en SQL del usuario, las turna

al RDS y despliega los resultados. Pueden escribirse programas de aplicaciones que utilicen directamente el RDS.

Compilación de consultas

Los usuarios comerciales de sistemas de base de datos cuentan con una serie de consultas que se ejecutan en forma periódica para generar reportes. Para evitar la necesidad de elegir una estrategia de procesamiento de consultas cada vez que se hace una de éstas, el Sistema R permite la *precompilación* de consultas. La precompilación consiste del análisis sintáctico de la consulta y de la elección de una estrategia completa para procesarla. El resultado de la precompilación se conoce como *plan*. Los planes se almacenan en la base de datos del Sistema R y se leen cada vez que se ejecuta la consulta.

Dado que un plan incluye detalles tales como la elección de estrategias de unión y la utilización de índices específicos, es posible que un plan deje de ser válido. Por ejemplo, podría darse el caso de que se borrara un índice utilizado por el plan. En estas situaciones es necesario volver a ejecutar la compilación de la consulta antes de que pueda ejecutarse ésta.

Lenguajes huésped

Cuando se presentaron los modelos de datos de red y jerárquico en los capítulos 4 y 5, se vio la necesidad de incrustar las solicitudes de acceso al sistema de base de datos dentro de un lenguaje de programación huésped. El modelo relacional no requiere de un lenguaje de consulta que incluya iteraciones, recursión o apuntadores, por lo que no es necesario incrustar los lenguajes relacionales que se presentaron en el capítulo 3 dentro de un lenguaje huésped. No obstante, la posibilidad de hacerlo muchas veces es útil en las aplicaciones comerciales de base de datos. Por ejemplo, un programador podría querer desarrollar una interfaz de aplicación especial con la base de datos para un grupo específico de usuarios. Para facilitar el desarrollo de tales aplicaciones, el Sistema R incluye al *SQL incrustado* (*embedded SQL*), que es una forma con ligeras modificaciones del SQL y que puede utilizarse dentro de un lenguaje de programación huésped. El Sistema R permite dos lenguajes huésped: PL/1 y Cobol. En el análisis que sigue se hablará sólo del PL/1.

Los problemas principales que implica la inclusión de los conceptos de lenguaje del SQL en el PL/1 surgen del hecho de que éste opera sobre registros, mientras que el SQL opera sobre conjuntos de registros (relaciones). Por tanto, se necesita un mecanismo para presentar al programa en PL/1 el resultado de una consulta en SQL (es decir, una relación) tupla por tupla (o sea, registro por registro). Para cada una de las relaciones que va a procesar el programa en PL/1 se define un *cursor*. Éste es un apuntador a una tupla y tiene una función similar a la de los apuntadores de actualidad del lenguaje de manipulación de datos DBTG. A continuación se resume el método de acceso normal de un programa en PL/1 a la base de datos:

- Se llama a un procedimiento que hace que se ejecute la consulta en SQL.

- Se abre un *cursor* para la relación que resulta de la ejecución de la consulta. Este cursor se utiliza para procesar una por una las tuplas de la relación resultante.
- Se hace una llamada de *búsqueda* (*fetch*) para obtener la "siguiente" tupla. La búsqueda obtiene la primera tupla de una relación asociada con un cursor que se acabe de abrir. Las siguientes llamadas de búsqueda avanzan al cursor hasta la siguiente tupla de la relación y leen la tupla a la que apunta el cursor. La tupla así obtenida se coloca en un registro de PL/1 para que el programa en PL/1 la pueda manipular.
- Se repiten las llamadas de búsqueda hasta que se hayan procesado todas las tuplas. Cuando esto sucede, la llamada de búsqueda produce un código especial.

Un programa real es ligeramente más complejo, debido a que:

- Es necesario que un preprocesador traduzca las proposiciones en SQL antes de que el compilador de PL/1 compile el programa. Se requieren comandos especiales para el preprocesador.
- Debe incluirse un procedimiento para transferir los valores de una tupla a variables de PL/1.
- Es posible que el tamaño de las tuplas que obtenga la búsqueda dependa de entradas que se hacen al programa. Esto sucede, por ejemplo, cuando el usuario introduce una consulta en SQL al programa mientras se está ejecutando.

En un programa real en PL/1 con SQL incrustado, aparece una cláusula adicional en las proposiciones en SQL. Ésta, la cláusula **into** ("a"), especifica las variables de PL/1 en las que se debe colocar el valor de cada uno de los atributos de la cláusula **select**. Para distinguir los nombres de variables de PL/1 de los nombres de relaciones y atributos, se les agrega un signo de dólar como prefijo siempre que aparecen en una proposición de SQL. Las variables de PL/1 pueden utilizarse también para efectuar comparaciones en la cláusula **where**.

Por ejemplo, considérese la consulta "encontrar los nombres de los clientes que tienen más de x dólares en alguna cuenta, y las ciudades donde viven", donde x es una variable del programa en PL/1. Supóngase que se utilizan las variables de PL/1 a y b para guardar una tupla del resultado. La consulta se escribe así:

```
select nombre-cliente, ciudad-cliente
  into $a, $b
  from depósito, cliente
  where depósito.nombre-cliente = cliente.nombre-cliente
        y depósito.saldo > $x
```

Los comandos para el preprocesador del Sistema R comienzan con el signo de dólar. Existen cuatro comandos:

- **$declare.** Este comando es idéntico al comando *declare* (declarar) del PL/1 y sirve para definir a las variables de PL/1 que se utilizan también en consultas en SQL.
- **$let** <nombre-cursor> **be** <proposición en SQL>. Este comando asocia a la proposición en SQL con el nombre del cursor. El procesador almacena esta asociación para utilizarla después. El comando **$let** *no* hace que se evalúe la consulta.
- **$open** <nombre-cursor>. Este comando hace que el preprocesador genere una llamada de PL/1 a un procedimiento que evalúa la consulta en SQL que está asociada al cursor. La asociación entre un nombre de cursor y una consulta en SQL debe haberse definido con anterioridad por medio de un comando **$let**. En este punto se produce una versión precompilada de la proposición en SQL. Cabe hacer notar que la proposición **into** no influye en este punto sobre el procesamiento de la consulta en SQL.
- **$fetch** <nombre-cursor>. Este comando hace que se coloquen los valores de la siguiente tupla de la relación asociada al cursor que está abierto en las variables de PL/1 que se listan en la cláusula **into**.

Para ilustrar los comandos anteriores se presentará un programa que imprime los resultados del ejemplo anterior, "encontrar los nombres de los clientes que tienen más de x dólares en alguna cuenta, y las ciudades donde viven":

```
$declare a char (20);
$declare b char (20);
$declare x binary fixed (15);
declare rc binary fixed (15);
$let c be select nombre-cliente, ciudad-cliente
          into $a, $b
          from depósito, cliente
          where depósito.nombre-cliente = cliente.nombre-cliente
                y depósito.saldo > $x
$open c;
rc = 0;
do while rc = 0;
     $fetch c;
     put skip list (a, b);
end
```

El procesador invierte el programa en uno susceptible de ser compilado por el compilador de PL/1. También precompila la consulta en SQL y almacena el plan correspondiente en la base de datos.

El ejemplo anterior se simplificó debido a que el programador de aplicaciones sabía que la consulta en SQL generaba una relación en *nombre-cliente* y *ciudad-cliente*. Esto permitió declarar las variables a y b del tipo de dato correcto. Considérese ahora el problema al que se enfrenta un programador que implanta la interfaz amable con el usuario (UFI). El programador no sabe por adelantado cuáles son las proposiciones en SQL que se van a introducir. De hecho, el programador no sabe siquiera cuáles son las relaciones que van a existir cuando se ejecute el programa de UFI.

Para poder crear programas que acepten consultas en SQL arbitrarias, el Sistema R incluye un comando **describe** (describir). Este comando acepta como entrada una proposición en SQL, y su salida es:

- El número de atributos de la relación resultante.
- El tipo de dato de cada uno de los atributos de la relación resultante.

El programador de aplicaciones debe asignar en forma dinámica las variables para aceptar las tuplas de la relación resultante.

Consistencia y concurrencia

Los conceptos de colocación de candados de dos fases y de granularidad múltiple se originaron en el proyecto del Sistema R. En el Sistema R todas las transacciones mantienen por omisión todos los candados hasta que se termina de ejecutar la transacción para garantizar la serialibilidad y evitar los retrocesos en cascada. Sin embargo, también ofrece la alternativa de *grados de consistencia* que permite liberar candados antes del término de la transacción. Un programador de aplicaciones que esté diseñando una transacción para el Sistema R, puede optar por un grado más débil de consistencia, ya que al liberar pronto los candados se reduce la probabilidad de que otra transacción tenga que esperar para colocar un candado.

El protocolo de candados de granularidad múltiple ha trabajado en forma satisfactoria en el Sistema R. Sin embargo, se hizo una pequeña modificación al esquema que se presentó en el capítulo 11 para resolver el siguiente problema: a menudo sucede que una transacción acumula un gran número de candados de granularidad de registro pero no coloca un candado de granularidad de relación. A resultas de ésto, el manejador de candados tiene que mantener una tabla de candados muy grande, y el tiempo extra por mantenimiento de candados aumenta. Para enfrentarse a este tipo de situaciones, el Sistema R efectúa en forma automática una *elevación de candados*. Siempre que sea posible, los candados de granularidad de registro se intercambiarán por un candado de granularidad de relación o de archivo. Este procedimiento de elevación de candados es transparente para el programador de aplicaciones y para el usuario.

El manejador de recuperaciones del Sistema R utiliza *tanto* la doble paginación como las bitácoras. Esta estrategia combinada refleja la evolución del Sistema R desde un sistema para un solo usuario (en el que la doble paginación funciona de manera satisfactoria) hasta uno de multiusuarios. La doble pagina-

ción no se adapta bien al caso de acceso concurrente a la base de datos. El sistema de bitácoras que se utiliza se denomina *bitácora de grabación adelantada* (*write-ahead logging*) y es en esencia el mismo que el de bitácora incremental con actualización inmediata que se describió en el capítulo 10.

Comentarios

El Sistema R se encuentra bien documentado en la literatura pública, debido a que es un proyecto de investigación y no un producto comercial. Aparte de su influencia sobre varios productos comerciales, el Sistema R ha dado pie a varios proyectos de investigación subsecuentes, incluyendo el proyecto de Sistema R* de base de datos distribuida y otras investigaciones que buscan extender el Sistema R para manejar objetos complejos (véase el Cap. 14).

15.1.2 SQL/Data System

El SQL/Data System (SQL/DS) es un sistema comercial de base de datos vendido por IBM; es de tamaño intermedio cuyo alcance se encuentra entre el de los sistemas de base de datos para macrocomputadoras (p. ej. DB2 e IMS) y los de base de datos para computadoras personales (p. ej., dBase III). El SQL/DS funciona con los sistemas operativos DOS/VSE o VM/SP, ambos de IBM.

El SQL/DS y el Sistema R tienen muchos puntos en común, ya que el primero se vio muy influenciado por el proyecto de investigación del Sistema R. Entre sus similaridades pueden mencionarse:

- El lenguaje de consulta SQL y el SQL incrustado.
- La precompilación.
- El control de concurrencia y manejo de transacciones.

Casi todo lo que se dijo en la sección 15.1.1 sobre estos temas se puede aplicar también al SQL/DS.

Un área importante en la que el SQL/DS va más allá del Sistema R es la capacidad para extraer información de las bases de datos IMS utilizando el lenguaje de consulta DL/1 de IMS. La función de extracción de DL/1 permite copiar datos de una base de datos IMS (jerárquica) a una parte de una base de datos SQL/DS (relacional). Una vez almacenada la información en el sistema SQL/DS, puede consultarse al utilizar el SQL. Sin embargo, no es posible modificar la base de datos IMS utilizando el SQL.

El lenguaje de definición de datos del SQL/DS incluye las proposiciones comunes del SQL: **create table** (crear tabla), **create index** (crear índice), **create view** (crear vista), etc. Además, el SQL/DS cuenta con proposiciones que permiten al administrador de la base de datos controlar la estructura física de la base de datos.

Un *espacio de base de datos* (*dbspace*, en la terminología del SQL/DS), es una sección de almacenamiento físico en disco en la que se almacenan las relaciones y sus índices. El mecanismo de seguridad del SQL/DS sirve para controlar la autoridad para agregar nuevas relaciones a un espacio de base de datos. Por ejemplo, puede asignarse un espacio de base de datos personal a un usuario para almacenar sus propias relaciones. A este usuario se le asignará autorización de recursos sobre su espacio de base de datos personal.

Cuando se crea un espacio de base de datos, puede asociarse con un disco específico. El poder asignar espacios de base de datos a discos específicos en un sistema de cómputo de discos múltiples permite al administrador de base de datos distribuir los accesos a información entre varios dispositivos de disco. Esta distribución de la carga de trabajo por lo general mejora el rendimiento.

La interfaz con el usuario del SQL/DS incluye tanto un precompilador como una interfaz interactiva similar a la UFI del Sistema R. La sintaxis del precompilador es diferente de la del Sistema R, pero las construcciones y conceptos básicos del lenguaje son los mismos.

La interfaz *interactiva SQL* (ISQL) permite al usuario introducir de manera directa consultas en SQL. Además de las proposiciones de SQL, la ISQL incluye proposiciones que controlan el formato de salida de las respuestas a las consultas, comandos para grabar y editar las consultas, y proposiciones para facilitar la generación de reportes.

15.1.3 Database 2

Database 2 (DB2) es un sistema comercial de base de datos vendido por IBM para utilizarse en macrocomputadoras grandes que tienen el sistema operativo MVS.

Aunque el DB2 y el SQL/DS son distintos, existe un gran parecido entre estos dos sistemas; ambos utilizan versiones casi idénticas del lenguaje de consulta SQL, y la sintaxis del SQL incrustado es en realidad la misma. Los dos sistemas no pueden compartir datos en forma directa, pero es posible transferir datos entre ellos.

Los conceptos fundamentales del DB2 son similares a los del SQL/DS y, por tanto, similares a los del Sistema R. La importancia del DB2 radica en que cuenta con el tamaño y grado de sofisticación adecuados para manejar bases de datos muy grandes. Por ello, y a diferencia del SQL/DS, el DB2 es un competidor del IMS para el manejo de bases de datos grandes. La introducción del DB2 en 1983 subrayó la creciente importancia del modelo de datos relacional en el ambiente comercial y el decaimiento en la importancia del modelo jerárquico. Las funciones DXT del DB2 permiten transferir información del IMS al DB2 (en forma análoga a la función de extracción de DL/1 del SQL/DS).

Gran parte de lo que se dijo acerca del Sistema R y del SQL/DS puede aplicarse al DB2. A continuación se mencionan algunos aspectos interesantes del DB2 de los que no se había hablado antes.

El DB2 interactúa con tres subsistemas diferentes que funcionan con el MVS. Cada uno de éstos permite un tipo distinto de manejo de transacciones. Estos tres sistemas, IMS, TSO (la opción de tiempo compartido del MVS) y CICS (sistema de control de información a clientes, en inglés, Customer Information Control System) son anteriores al DB2. Por ello, la interacción del DB2 con estos subsistemas fue una necesidad práctica. Con todo, da pie a una aplicación inte-

resante de los conceptos de comisión de transacciones en bases de datos *distribuidas*. En el capítulo 12 se supuso que un sistema distribuido consiste de computadoras conectadas por medio de una red. Dentro del DB2, los subsistemas coexisten dentro de la misma máquina. Sin embargo, como en el caso de los sistemas distribuidos, es preciso que los subsistemas estén de acuerdo, ya sea para cometer o para abortar las transacciones que utilicen los servicios de más de un subsistema. Esto se logra dentro del DB2 al utilizar el protocolo de comisión de dos fases que se vio en el capítulo 12.

La *facilidad de manejo de consultas* (QMF, *query management facility*) del DB2 es una interfaz que permite al DB2 interactuar con la base de datos al emplear:

- SQL.
- QBE.
- Un generador de informes.

El lenguaje QBE se describió en el capítulo 3. En un principio, el QBE era un sistema de base de datos completo desarrollado en el IBM T. J. Watson Research Center. La característica más notable del sistema QBE es su lenguaje de consultas. Dado que tanto el QBE como el SQL son lenguajes de consulta relacionales, es factible que un sistema de base de datos ofrezca ambos a sus usuarios.

Al utilizar la QMF, los usuarios de DB2 pueden compartir información aun en el caso en que uno utilice SQL y otro QBE. Los usuarios de ambos lenguajes pueden disponer también del generador de informes. La QMF genera en forma automática un formato para el reporte, y el usuario puede alterar este formato si lo desea. En la actualidad, también es posible que el SQL/DS aproveche la función QMF.

15.1.4 Oracle

El sistema de base de datos Oracle, vendido por la Oracle Corporation, es un sistema que aunque no fue desarrollado por la IBM, se apega fielmente al lenguaje de consultas y a la interfaz con el usuario del producto SQL/DS de IBM.

Oracle funciona en muchos sistemas diferentes, incluyendo las computadoras DEC VAX y las similares a la IBM-370. Como los sistemas IBM antes descritos, Oracle utiliza el lenguaje de consulta SQL y permite que los programas de aplicaciones utilicen el SQL incrustado. Se ofrece SQL incrustado para los lenguajes Cobol, Fortran y C. Además de una técnica de precompilador para manejar el SQL incrustrado, Oracle incluye una *interfaz de llamadas* que interpreta las proposiciones de SQL en el momento de la ejecución, en vez de compilarlas. Las llamadas de la interfaz de llamadas corresponden directamente a las proposiciones del precompilador. Aunque la interfaz de llamadas elimina el paso de precompilación cuando se prepra un programa de aplicaciones, es preferible utilizar la técnica de mpilación para los programas que se van a ejecutar varias veces. Como ca dos los sistemas comerciales, Oracle incluye un generador de informes. Sin embargo, las características de este generador son muy diferentes

de las de los sistemas que ya se mencionaron (excepto por el hecho de utilizar todos el SQL).

Oracle utiliza índices de árbol B^+ con una opción de *compresión de llaves*. La compresión de llaves es una técnica para ahorrar espacio. En vez de guardar la llave completa en el índice, sólo se guarda una parte que baste para identificar el apuntador que es preciso seguir. Esta técnica ahorra una cantidad considerable de espacio cuando las llaves son largas (como en el caso de nombres de personas), pero tiene la desventaja de que para determinar el valor completo de una de ellas es preciso leer un registro de la base de datos.

La recuperación de caídas del Oracle se basa en *archivos de imagen previa*. El archivo de imagen previa contiene copias de los bloques de la base de datos que hayan sido modificados por las transacciones sin cometer. Estas copias se usan para el retroceso de las transacciones abortadas. La técnica es similar a la de doble paginación que se presentó en el capítulo 10.

El control de concurrencia de Oracle utiliza candados y permite un grado todavía mayor de concurrencia por medio del uso del archivo de imagen previa. Esta técnica es una forma de control de concurrencia de multiversión en el que sólo se permiten dos versiones. Supóngase que la transacción T_1 solicita acceso para lectura a los datos que T_2 está modificando. Con los protocolos para candados normales, T_1 tendría que esperar a que T_2 libere el candado. Oracle, por el contrario, permite a T_1 leer los valores previos a la modificación que realiza T_2. Estos valores están almacenados en el archivo de imagen previa. Si no se desean los datos de la imagen previa, T_1 puede solicitar un candado, lo que la obligará a esperar a que cualquier operación de escritura que se esté efectuando en forma concurrente libere la información.

15.1.5 Ingres

En la misma época en que el IBM San José Research Laboratory estaba desarrollando el prototipo del Sistema R, un grupo de investigadores en la University of California at Berkeley desarrollaba un sistema experimental de base de datos llamado Ingres. Aunque ambos proyectos partieron de los conceptos del modelo de datos relacional, presentan diferencias sustanciales en cuanto al diseño del sistema y su interfaz con el usuario.

El proyecto de investigación Ingres condujo al desarrollo de un producto comercial del mismo nombre, vendido por Relational Technology, Inc. En esta sección se analizará la versión académica de Ingres, aunque muchos de los comentarios pueden aplicarse también a la versión comercial.

Estructura general del sistema

Ingres se desarrolló en una computadora PDP-11 con el sistema operativo UNIX. Este sistema operativo tuvo una influencia considerable sobre la estructura interna de Ingres. Se utilizan varios procesos, cada uno de los cuales se encarga de una tarea específica:

- Formulación de consultas, interacción con el usuario.

- Análisis lexicológico y sintáctico, optimización de consultas.
- Ejecución de consultas.
- Mantenimiento de índices.

La solicitud que hace el usuario se turna de proceso a proceso según sea necesario.

Este diseño de multiprocesos permite la concurrencia dentro de Ingres mismo, ya que es posible, por ejemplo, efectuar el análisis sintáctico de una consulta mientras se ejecuta otra. Sin embargo, la comunicación interproceso (por medio de los conductos Unix) conlleva un cierto tiempo extra. Históricamente, uno de los factores que más influenciaron el diseño de Ingres fue la limitación que tenía la PDP-11 de espacios de direccionamiento de 64 kilobytes para cada proceso individual.

Ingres introdujo el lenguaje de consulta Quel y una incrustación de éste dentro del lenguaje huésped C llamada Equel. Ya se habló del Quel en el capítulo 3. Gran parte de la sintaxis de Equel tiene una forma similar a la de SQL incrustado. Se utilizan los caracteres "##" para distinguir a una proposición de Quel que aparece dentro de un programa en C. Los caracteres "##" se emplean también para marcar las declaraciones en C que se refieran a variables a las que se hace referencia dentro de una proposición en Quel.

Equel difiere de SQL incrustado en su equivalente de los cursores de SQL. Recuérdese que en el SQL incrustado, se *abre* un cursor para una consulta y a continuación se obtienen tuplas individuales por medio de una llamada de búsqueda (**fetch**). El Equel utiliza un enfoque más estructurado en el que se ejecuta una sección específica del código, una vez por cada tupla del resultado de una consulta en Quel. La figura 15.1 muestra un ejemplo de programa en Equel que imprime el nombre y la ciudad de los clientes que tienen más de x dólares en

```
## char a [20], b [20];
## float x;
main () {

## range of d is depósito
## range of c is cliente
## retrieve a = c.nombre-cliente, b = c.ciudad-cliente
## where d.saldo > x
    # { printf ("%5, %7.2f\n", a, b)
    # }
```

Figura 15.1 Ejemplo de programa en Equel.

una cuenta. El código delimitado por "#{" y "#}" se ejecuta una vez por cada tupla del resultado. Los valores de la tupla para cada uno de los atributos de la relación resultante se almacenan en las variables de la cláusula **retrieve** (obtener) (a y b en la figura 15.1).

Ingres no permite la precompilación de las consultas en Quel, aun en el caso en que la consulta está incrustada en un programa en C (Equel). La optimización de consultas se realiza como parte de la interpretación de éstas, en vez de hacerse durante la precompilación. Una técnica de optimización de consultas introducida por Ingres, y que ha resultado muy útil, se denomina *descomposición*. Se construye una *gráfica de consultas* cuyos nodos son relaciones y cuyas aristas representan productos theta o naturales. Si una arista representa el producto de una relación pequeña con otra, el resultado esperado es otra relación pequeña. Por tanto, los productos de este tipo se procesan primero. Cuando no queden más de estos productos, se lleva a cabo la descomposición.

Para ilustrar la descomposición se utilizará la parte de una gráfica de consultas que se muestra en la figura 15.2. Esta gráfica ilustra el producto de tres relaciones, r_1, r_2, y r_3. Los esquemas correspondientes son R_1, R_2 y R_3. La relación r_2 se descompone de la siguiente manera:

- Se construye un índice por r_1 para $R_1 \cap R_2$ (a menos que ya exista ese índice).
- Se construye un índice por r_3 para $R_3 \cap R_2$ (a menos que ya exista ese índice).
- Se descompone r_2 en varias relaciones. Cada relación tiene el esquema R_2 y contiene sólo una tupla de r_2. Se produce una gráfica de consulta para cada una de las relaciones descompuestas. En cada una de estas gráficas una relación descompuesta toma el lugar de r_2.
- Se procesa cada una de las gráficas de consulta y se calcula la unión de los resultados.

Puesto que todas las gráficas de consulta que se generan al aplicar la descomposición tienen la misma estructura, es fácil procesarlas.

15.2 Sistemas de red

El modelo de red es la base de la mayor parte de los sistemas antiguos de base de datos, y de algunos sistemas recientes. En esta sección se analizarán brevemente dos sistemas de red populares, Total e IDMS.

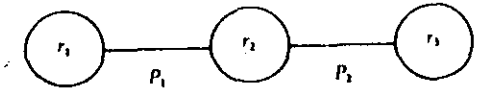


Figura 15.2 Parte de una gráfica de consultas de Ingres.

15.2.1 Total

El sistema de base de datos Total data de finales de la década de 1960. Funciona en una gran variedad de máquinas que van desde minicomputadoras hasta macrocomputadoras.

Aunque Total se basa en el modelo DBTG, su lenguaje de consulta difiere del que se presentó en el capítulo 4. El lenguaje presentado en el capítulo 4 está basado en el estándar DBTG. El lenguaje de Total, aunque tiene una función similar, utiliza una sintaxis diferente.

Todas las proposiciones del lenguaje de manipulación de datos Total deben incrustarse en un lenguaje huésped (Cobol, PL/1, Fortran o RPG). Se hace una llamada al procedimiento *database* con parámetros que especifican la proposición en lenguaje de manipulación de datos. Estos comandos son similares a los comandos **find** (encontrar), **find owner** (encontrar dueño), y **find next** (encontrar el siguiente) que se vieron en el capítulo 4, aunque la terminología que utiliza Total tiene su origen en algunos de los conceptos del lenguaje DL/1 del IMS. Los tipos dueños se denominan conjuntos de datos de entrada única y los tipos miembros se llaman conjuntos de datos de entrada variable.

La implantación física interna de Total y los esquemas de acceso que se utilizan no han sido publicados.

15.2.2 IDMS

IDMS es un sistema de base de datos de red desarrollado por Cullinane Database Systems, Inc. IDMS sigue de cerca el modelo DBTG tal y como se presentó en el capítulo 4; incluye un lenguaje de manipulación de datos detallado que permite al diseñador de base de datos un grado de control considerable sobre la organización física de la base de datos. El lenguaje de manipulación de datos tiene características idénticas (o casi idénticas) al lenguaje de manipulación de datos DBTG (DBTG DML) que se describió en el capítulo 4. Se han incluido comandos adicionales para facilitar la labor del programador y para que los programadores experimentados sean capaces de redactar consultas más eficientes.

Un ejemplo de las características de IDMS que no se mencionaron en el capítulo 4 es el comando **obtain** (obtener) que combina en una sola solicitud los comandos **find** y **get**. Puede agregarse una cláusula **where** opcional al comando **obtain** para localizar y obtener el siguiente registro que satisfaga el predicado de la cláusula **where**. Esto le ahorra al programador la tarea de escribir una prueba explícita para un registro que se haya localizado por medio del comando **find**.

Las notas bibliográficas hacen referencia a algunos documentos que describen en forma detallada el manejo interno de archivos y las técnicas de indización empleadas por el IDMS.

15.3 Sistemas jerárquicos

El modelo je ico tiene relevancia en primer término debido a la importancia del sistema se de datos IMS de IBM. En esta sección se hablará tanto del IMS como de otro sistema jerárquico muy utilizado, el Sistema 2 000.

15.3.1 IMS

El sistema de Manejo de Información (IMS, Information Management System) de IBM es uno de los sistemas de base de datos más antiguos y más utilizados. Trabaja con el sistema operativo MVS en máquinas grandes IBM-370 y similares. Por tradición, las bases de datos más grandes han sido bases de datos IMS, por lo que las personas encargadas del desarrollo de este sistema fueron de las primeras en tener que enfrentarse a los problemas de concurrencia, recuperación, integridad y procesamiento eficiente de las consultas. Al pasar por varias versiones, IMS fue adquiriendo un gran número de funciones y opciones y, por esta razón, es un sistema de gran complejidad; aquí se analizarán sólo algunas de sus características.

IMS está basado en el modelo de datos jerárquico (véase el Cap. 5). Las consultas a las bases de datos IMS se hacen por medio de llamadas incrustadas en un lenguaje huésped. Las llamadas incrustadas son parte del lenguaje de base de datos DL/1 del IMS. El lenguaje que se utilizó en el capítulo 5 es una forma simplificada de DL/1.

Puesto que el rendimiento es de primordial importancia en las bases de grandes datos, IMS permite al diseñador de base de datos un gran número de opciones dentro del lenguaje de definición de datos. el diseñador define al esquema de la base de datos como una jerarquía física. Pueden definirse varios subesquemas (o vistas) construyendo una jerarquía lógica a partir de los tipos de registro que constituyen el esquema. El lenguaje de definición de datos incluye una gran variedad de opciones (tamaños de bloques, campos apuntadores especiales, etc.) que permiten al administrador de la base de datos "afinar" al sistema con el fin de mejorar su rendimiento.

IMS cuenta con varios esquemas de acceso a registros:

- HSAM (método jerárquico de acceso secuencial, *hierarchical sequential-access method*), que se utiliza para archivos cuya organización física es secuencial (como los archivos en cinta). Los registros se almacenan físicamente en preorden.
- HISAM (método jerárquico de acceso secuencial indizado, *hierarchical indexsequential access method*), que es una organización secuencial indizada en el nivel de raíz de la jerarquía.
- HIDAM (método jerárquico indizado de acceso directo, *hierarchical indexed direct-access method*), que es una organización indizada en el nivel de la raíz con apuntadores a los registros hijos.
- HDAM (método jerárquico de acceso directo, *hierarchical direct-access method*), que es similar al HIDAM pero con acceso por cálculo de dirección al nivel de la raíz.

La versión original del IMS antecede al desarrollo de la teoría ntrol de concurrencia. Las primeras versiones del IMS contaban con una fo a sencilla de control de concurrencia: sólo podía ejecutarse un programa que incluyera ac-

tualizaciones a la vez. Sin embargo, podían ejecutarse de manera concurrente cualquier cantidad de aplicaciones de lectura y una aplicación de actualización. Esto permitía a las aplicaciones leer valores actualizados no cometidos y ejecutarse en forma no serializable. La única opción disponible para las aplicaciones que requerían mayor protección contra las anomalías del procesamiento concurrente era el acceso exclusivo a la base de datos.

Versiones posteriores del IMS incluyeron una *función de aislamiento de programas* más complejos, que permitía un mejor control de la concurrencia y técnicas de recuperación de transacciones, a su vez más complejas (p. ej., las bitácoras). Estas características se hicieron cada vez más importantes conforme se popularizó el uso de transacciones en línea en vez de las transacciones por lote que predominaban en un principio.

La necesidad de un procesamiento de transacciones de alto rendimiento condujo a la introducción del *IMS Fast Path* (ruta rápida). El Fast Path utiliza una alternativa de organización física diseñada de tal manera que permite que las partes más activas de la base de datos residan en la memoria principal. En vez de forzar la salida de actualizaciones al disco después de una transacción (como lo hace el IMS estándar), la actualización se posterga hasta el siguiente punto de verificación o de sincronización. En el caso de una caída, el subsistema de recuperación debe repetir todas las transacciones cometidas cuyas actualizaciones no se grabaron en disco. Estos y otros "trucos" permiten una velocidad de procesamiento de transacciones muy alta.

Una base de datos IMS puede consultarse, pero no actualizarse, utilizando el SQL. La función DXT del DB2 y la función de extracción de DL/1 del SQL/DS permiten almacenar datos de IMS en una base de datos relacional en la que pueden realizarse consultas en SQL.

Los detalles completos del IMS están más allá del alcance de este breve panorama. Aunque el IMS es un sistema antiguo y la importancia del modelo jerárquico disminuye, la historia y la evolución del IMS constituyen un marco de referencia interesante para el estudio del desarrollo de los conceptos de sistemas de base de datos.

15.3.2 Sistema 2 000

El Sistema 2 000 es un sistema jerárquico de base de datos desarrollado en un principio por la MRI Corporation y distribuido en la actualidad por Intel. El Sistema 2 000 puede utilizarse en computadoras tipo IBM-370, Univac 1 100 y en CDC 6 000 y Cyber. Aunque utiliza el mismo modelo de datos que el IMS, las características del lenguaje que ofrece el Sistema 2 000 presentan varias diferencias interesantes con respecto al lenguaje DL/1 el IMS.

El concepto fundamental del lenguaje de definición de datos DL/1 es la relación padre hijo uno a muchos. En el ejemplo bancario, supóngase que existe una relación uno a muchos entre *cliente* y *cuenta*. En el diseño de la base de datos, *cuenta* se definiría como hijo de *cliente*. Otra forma de considerar esta relación uno a muchos es, como un registro que tiene muchos campos repetidos. Dentro del registro *cliente* de un cliente determinado puede almacenarse el conjunto de cuentas que pertenecen a ese cliente si se crea un campo *cuenta* repetido. Este enfoque de campos repetidos es el fundamento del lenguaje de definición de datos del Sistema 2 000.

En el ejemplo bancario del capítulo 5, el registro *cliente* tiene los campos *nombre-cliente*, *calle* y *ciudad-cliente*, y el registro *cuenta* tiene los campos *número-cuenta* y *saldo*. A continuación se muestran las proposiciones que corresponden a esta parte del sistema de base de datos utilizando el lenguaje de definición de datos del Sistema 2 000:

```
1* nombre-cliente (name);
2* calle (name);
3* ciudad-cliente (name);
4* cuenta (repeating group);
5* número-cuenta (integer);
6* saldo (money);
```

Los tipos de datos *name* (nombre), *integer* (entero) y *money* (dinero), entre otros, vienen incluidos en el Sistema 2 000. Los campos se especifican como *key* (llave) o *nonkey* (no llave). El Sistema 2 000 construirá un índice por cada uno de los campos llave.

Considérese la consulta "encontrar los nombres de todos los clientes que tienen una cuenta cuyo saldo es superior a $10 000, y las ciudades donde viven". Si se utiliza el lenguaje similar al DL/1 del capítulo 5, es preciso revisar todos los registros de cuentas de cada cliente, y para ello se escribe un ciclo **while** en el lenguaje huésped. El lenguaje de consultas del Sistema 2 000, QUEST, permite escribir la consulta utilizando una sola proposición.

```
list nombre-cliente, ciudad-cliente
while cliente has saldo > 10 000
```

También es posible incrustar las consultas del Sistema 2 000 en un lenguaje huésped. Un precompilador procesa el programa. El análisis sintáctico de las consultas se hace durante la fase de precompilación. Al igual que la mayor parte de los sistemas comerciales, el Sistema 2 000 incluye funciones que auxilian en la generación de reportes.

15.4 Sistemas de bases de datos para microcomputadoras

La característica fundamental de los sistemas de base de datos para microcomputadoras, es su sencillez. La capacidad limitada de las computadoras personales restringe tanto el tamaño de la base de datos como el grado de complejidad del sistema. Aunque casi todos los sistemas de base de datos se diseñan pensando en la facilidad de uso, la importancia de este factor en el mercado de las computadoras personales es extraordinaria, ya que los usuarios no pueden contar con

la ayuda de un administrador de base de datos experimentado. Cada uno de los usuarios de un sistema de base de datos de microcomputadora funge como administrador de base de datos.

A continuación se compararán las características comunes de los sistemas de base de datos para microcomputadoras con las de los sistemas más grandes:

- **Modelo de datos.** Dado que los sistemas de base de datos para microcomputadoras son relativamente nuevos, casi todos están basados en el modelo relacional. Algunos de esos sistemas deben considerarse más bien *tabulares*, ya que, aunque utilizan tablas, son demasiado primitivos para llamarse relacionales.
- **Lenguaje de consultas.** Es posible que aun los lenguajes de más alto nivel que se analizaron en este texto sean demasiado complejos para el usuario casual de un sistema de base de datos para microcomputadora. Muchos de los lenguajes se basan en una interfaz de forma, con la cual el usuario puede interactuar con el sistema llenando una forma.
- **Implantación física.** Para los implantadores de un sistema, uno de los factores importantes es el espacio que ocupa el código objeto de un sistema de base de datos para microcomputadora. Si se reduce el espacio requerido, es posible utilizar el sistema en máquinas que cuenten con menos memoria principal. Esto puede tener una influencia determinante sobre el mercado potencial. Por esta razón son pocos los sistemas que utilizan técnicas de manejo de memoria y de indización complejas. Por lo general se elige un solo tipo de índice, y la optimización de consultas, cuando se lleva a cabo, es rudimentaria.
- **Recuperación.** Muchos sistemas no cuentan con subsistema de recuperación. El usuario tiene la responsabilidad de sacar copias de respaldo de sus datos en forma regular.
- **Concurrencia.** No se requiere control de concurrencia para computadoras personales de un solo usuario.

La distinción entre los sistemas de base de datos para microcomputadoras y los sistemas más grandes se hace menos clara. Los primeros sistemas de base de datos para microcomputadoras no eran mucho más que interfaces para lograr acceso a un solo archivo de registros de longitud fija. Al crecer la capacidad de las computadoras personales, ha crecido también la complejidad de los sistemas de base de datos para microcomputadora. De hecho, han comenzado a aparecer versiones de algunos de los sistemas de base de datos de gran escala en las computadoras personales de mayor tamaño. Un ejemplo de este tipo de sistemas es el SQL/P[illegible]e la computadora personal IBM RT.

En esta secc[illegible] examinará el dBase-III, que ha tenido gran éxito en el mercado de los sistemas de base de datos para microcomputadora, y el Lotus 1-2-3, que es un sistema de "base de datos de hoja electrónica".

15.4.1 dBase-III

El dBase-III de Ashton-Tate es una actualización del exitoso producto comercial dBase-III. Incluye tanto un lenguaje de manipulación de datos como uno de programación de aplicación general. Aunque las características de estos lenguajes están basadas en el Cobol, Pascal y los de base de datos que se estudiaron aquí, el lenguaje del dBase-III es único. El lenguaje de programación es necesario para todas las consultas, con excepción de las más sencillas, debido a que el lenguaje de manipulación de datos, por sí solo, es menos poderoso que el álgebra relacional.

La selección y la proyección pueden expresarse en dBase-III al emplear el comando **display** (exhibir). Considérese por ejemplo la consulta: "encontrar los nombres de todas las sucursales en las que Jones tiene una cuenta". En primer término, se escribe la proposición:

use *depósito*

para indicar que se desea utilizar el archivo *depósito* que contiene la tabla *depósito*. La proposición:

display all off for *cliente* = "Jones", *nombre-sucursal*

muestra la respuesta a la consulta. La palabra clave **all** (todos) hace que se exhiban todos los registros que satisfacen a la consulta, no sólo uno de ellos. La palabra clave **off** suprime la impresión de números de registro.

La proposición **delete** (borrar) utiliza una sintaxis similar. Una característica interesante del borrado en dBase-III es que a los registros borrados sólo se les *marca* como tales, pero en realidad no se eliminan de la base de datos. Esto permite a un usuario nulificar un borrado erróneo por medio de la proposición **recall** (recordar). Los registros borrados se eliminan de la base de datos sólo cuando el usuario hace que se ejecute la proposición **pack** (empacar).

La proposición **append** (agregar) hace que el dBase-III entre al modo de inserción. En la pantalla se exhibe una forma con un espacio en blanco por cada campo de un registro de la tabla que se especifica en la proposición **use** (utilizar) más reciente. El usuario llena la forma.

El cálculo de productos en dBase es más laborioso que en los lenguajes de consulta que se describieron en el capítulo 3. El usuario debe crear una tabla nueva que contenga el producto. Se requiere un comando **select** (elegir) para poder especificar varias tablas en las proposiciones **use**. Cabe hacer notar que esta forma de utilizar la palabra **select** no corresponde al significado que tiene en SQL ni al de la selección en el álgebra relacional. La secuencia de proposiciones que se requiere para calcular:

depósito × *cliente*

es

select 2

```
use depósito
select 1
use cliente
join with depósito to cd for nombre-cliente = cliente — > nombre-cliente;
    fields nombre-sucursal, número-cuenta,
    cliente- > nombre-cliente, saldo, calle, ciudad-cliente
use cd
```

Las dos proposiciones **select** y **use** especifican las tablas a las que se va a tener acceso. Se realiza el producto de la tabla *cliente* con *depósito* en la proposición **join**, y el tipo de producto que se lleva a cabo es theta, seguido de una proyección. La cláusula **for** (para) de esta proposición especifica el predicado de producto. La cláusula **fields** (campos) especifica los atributos en los que se va a proyectar el producto theta. Nótese que el dBase-III utiliza la notación:

nombre-tabla — > nombre-atributo

en vez de la notación

nombre-relación.nombre-atributo

que se utilizó en los capítulos 3 y 6. El resultado del producto es una tabla nueva, *cd*, que contiene al producto, puede usarse la proposición *display* para exhibir el resultado.

Si son necesarios varios productos para responder a una consulta, y se sigue un método similar al antes descrito, será preciso crear un gran número de archivos temporales. A diferencia de los sistemas grandes de base de datos en los que el optimizador de consultas se encarga de calcular los productos de manera eficiente, el dBase obliga al usuario a escribir un programa para calcular en forma eficiente el producto de varias relaciones.

El lenguaje de programación del dBase incluye el acceso a la base de datos como parte integral del lenguaje, en vez de incrustarlo en un lenguaje ya existente por medio de caracteres de escape especiales (como el "$" del Sistema R o el "#" del Ingres). No se presentará aquí el lenguaje de programación de dBase-III completo. Tiene las estructuras de control usuales (**if-then-else, while, case**). Además, tiene un comando **find** (encontrar) que permite localizar los registros que tienen un valor determinado en el campo que se especifique. Sin embargo, este comando funciona sólo para los campos indizados. Para los demás campos es preciso emplear el comando **locate** (localizar) que realiza una búsqueda lineal. Una vez localizado el registro con cualquiera de los dos comandos, el programa puede realizar operaciones con los valores de los campos del registro.

Debido a la capacidad limitada del lenguaje de consulta del dBase, es necesario utilizar el lenguaje de programación del dBase con más frecuencia que el SQL incrustado del Sistema R. El programador de dBase debe estar consciente de la eficiencia de la estrategia de procesamiento de consultas que va a utilizar, mientras que el programador de SQL incrustado puede dejar la mayor parte de las consideraciones de eficiencia al optimizador de consultas.

	A	B	C	D	E
1	millas recorridas	precio (dlls/gal)	importe	galones	mi/gal
2				C2/B2	A2/D2
3				C3/B3	A3/D3
4				C4/B4	A4/D4
5				C5/B5	A5/D5

Figura 15.3 Hoja electrónica de rendimiento de gasolina-fórmulas.

15.4.2 Sistemas de base de datos de hoja electrónica

El Visicalc, producido por Software Arts, Inc., fue el primer lenguaje de hoja electrónica exitoso. A diferencia de los lenguajes de programación tradicionales, la naturaleza del lenguaje de hoja electrónica es bidimensional. Una hoja electrónica es un arreglo bidimensional de celdas. Para programar con una hoja electrónica es necesario definir relaciones matemáticas entre las celdas. El sistema de hoja electrónica mantiene estas relaciones al irse introduciendo los datos. Por tanto, a cada celda puede estar asociada una *fórmula* o un *valor*.

La figura 15.3 muestra las fórmulas de una hoja electrónica sencilla que mantiene los registros de rendimiento de gasolina de un automóvil. En una hoja de cálculo, las columnas reciben los nombres A, B, ..., y las hileras 1, 2, ..., . En la hoja aparecen las fórmulas para calcular el número de galones adquiridos y las millas recorridas por galón. Obsérvese, por ejemplo, la celda D2, que contiene la fórmula C2/B2. Esto quiere decir que el valor de D2 se calcula dividiendo el valor que está en la celda C2 entre el que está en B2. Supóngase que se introducen en la hoja los valores que se muestran en la figura 15.4. La hoja electrónica que ve el usuario se muestra en la figura 15.5. En vez de las fórmulas, el usuario ve el resultado de las mismas. Existen comandos especiales para exhibir y modificar las fórmulas.

El objetivo de esta sección no es describir en la forma detallada la programación de hojas electrónicas. Lo que se quiere hacer resaltar es que el conjunto rectangular de celdas de una hoja electrónica puede considerarse como una tabla. Esto sugiere la combinación de los lenguajes de hoja electrónica y de base de datos dentro de una sola estructura. Este tipo de lenguaje se ilustrará por medio del producto Lotus 1-2-3 de la Lotus Development Corporation.

El esquema de una tabla 1-2-3 se define al introducir el nombre de campo (atributo) para cada una de las columnas de la hilera 1. La hilera 1 es la única

	A	B	C	D	E
1	millas recorridas	precio (dlls/gal)	importe	galones	mi/gal
2	351	1.099	9.00		
3	292	1.119	8.25		
4	302	1.099	8.70		
5	289	1.079	9.10		

Figura 15.4 Hoja electrónica de rendimiento de gasolina: introducción de valores.

	A	B	C	D	E
1	millas recorridas	precio (dlls/gal)	importe	galones	mi/gal
2	351	1.099	9.00	8.19	42.0
3	292	1.119	8.25	7.37	39.6
4	302	1.099	8.70	7.92	38.0
5	289	1.079	9.10	8.43	34.3

Figura 15.5 Hoja electrónica de rendimiento de gasolina: resultado que ve el usuario.

que puede utilizarse para esto. No es preciso que el usuario se preocupe mucho por los tipos de datos. Todo lo que tiene que hacer es decidir el número de caracteres que va a necesitar para desplegar los valores en cada columna. Así, de hecho, todos los dominios son del tipo *cadena*, y su longitud la especifica el usuario. Los datos se introducen en la base de datos de hoja electrónica de la misma manera que se hace con los datos de una hoja de cálculo ordinaria.

Las tablas del Lotus 1-2-3 difieren de las hojas de cálculo sencillas en cuanto a que pueden efectuarse en ellas clasificaciones y búsquedas. El comando de clasificación (/DS) requiere como entradas la especificación de un rango continuo para clasificar y la llave de clasificación (las columnas según las cuales se van a clasificar las hileras).

Para obtener información de una tabla de Lotus 1-2-3 se utiliza el comando/DQ (data query, consulta de datos). Como en el caso del comando de clasificación, es preciso especificar un intervalo contiguo de hileras para la búsqueda. El predicado se especifica creando una plantilla en forma de tabla, donde los encabezados de las columnas son los nombres de los campos que deben tener un valor específico en el predicado. La forma de introducir el predicado a esta plantilla recuerda al QBE (consulta por ejemplo). La hilera número i de la plantilla especifica un predicado P_i. La consulta es:

$$P_1 \wedge P_2 \wedge \cdots$$

Cada P_i es una conjunción de condiciones referentes a campos individuales. Si aparece una constante C en la columna i de una hilera, sólo se obtendrán las hileras que contengan el valor C en la columna i. La figura 15.6 muestra la consulta, "encontrar todos los registros en los que el precio sea $1.099".

	A	B	C	D	E
1	millas recorridas	precio (dlls/gal)	importe	galones	mi/gal
2	351	1.099	9.00	8.19	42.0
3	292	1.119	8.25	7.37	39.6
4	302	1.099	8.70	7.92	38.0
5	28º	1.079	9.10	8.43	34.3
6		precio (dlls/gal)			
		1.099			

Figura 15.6 Consulta para localizar las hileras en las que el precio sea 1.099.

	A	B	C	D	E
1	millas recorridas	precio	importe	galones	mi/gal
2	351	1.099	9.00	8.19	42.0
3	292	1.119	8.25	7.37	39.6
4	302	1.099	8.70	7.92	38.0
5	289	1.079	9.10	8.43	34.3
6	millas recorridas	precio	importe	galones	mi/gal
					+ E2 > 40 *y* E2 < 50

Figura 15.7 Consulta para localizar las hileras en las que el rendimiento esté entre 40 y 50.

Para especificar condiciones que involucren comparaciones distintas de la igualdad pueden utilizarse los símbolos ">", "<", etc. No hay un equivalente directo de las variables de dominio del QBE, pero la colocación en la hoja de cálculo del valor de un campo en la primera hilera puede utilizarse como una forma de variable de dominio. La figura 15.7 muestra la consulta, "encontrar todos los registros en los que el rendimiento fue mayor que 40 y menor que 50 millas por galón".

En todas las consultas de 1-2-3, las hileras que forman el resultado se copian en un *rango* de salida, es decir, una sección de la hoja de cálculo que el usuario especifica con anterioridad como parámetro de un comando/DQ.

Sólo se mencionaron aquí algunas de las características del comando/DQ. No obstante, es necesario recalcar que aunque no es difícil expresar la mayor parte de las consultas básicas, el lenguaje de consulta no tiene el poder ni la amplitud de los lenguajes de consulta que se describieron en los capítulos 3, 4 y 5.

Notas bibliográficas

La distinción entre los sistemas de base de datos relacionales y tabulares se analiza en [Codd 1982].

En un gran número de trabajos publicados se presentan diversos aspectos del Sistema R. Astrahan et al. [1976], Astrahan et al. [1979], y Blasgen et al. [1979] presentan un panorama general del Sistema R. Chamberlin et al. [1976] introdujo el lenguaje SQL. Chamberlin [1980], Reisner [1977] y Reisner et al. [1975] presentan un análisis de los factores humanos del SQL. Eswaren et al. [1976], Gray [1978] y Gray et al. [1975, 1976] comentan el control de concurrencia en el Sistema R. Gray et al. [1981a] analiza la técnica de recuperación de caídas del Sistema R. Griffiths y Wade [1976], Fagin [1978], y Chamberlin et al. [1978] analizan la seguridad y las autorizaciones en el Sistema R. Lorie y Wade [1979] y Selinger et al. [1979] explican cómo se lleva a cabo la compilación de consultas en el Sistema R. Chamberlin et al. [1981] presentan un panorama del Sistema R que se escribió después del término del proyecto.

Williams et al. [1982] presenta un panorama general de R*, que es un[a] versi[ón] stribuida experimental del Sistema R. Lindsay et al. [1980] prese[nta un] pa[no]ama más detallado de la forma como el Sistema R* enfoca el manejo [de] tribuido de datos. Traiger et al. [1982] analizan la recuperación; Lindsay [1981] ha-