

Sistemas de Bases de Datos

Carlos González Alvarado



Editorial Tecnológica
de Costa Rica



Sistemas de Bases de Datos

Carlos González Alvarado



Primera edición.
Editorial Tecnológica de Costa Rica, 1996.

Primera reimpresión, 1998.
Segunda reimpresión, 2000.
Tercera reimpresión, 2002.
Cuarta reimpresión, 2009.

005.756

González Alvarado, Carlos Alberto
Sistemas de bases de datos.

ISBN 9977-66-097-2

I. Bases de datos. I. T.

© Editorial Tecnológica de Costa Rica
Correo electrónico: editorial@itcr.ac.cr
Instituto Tecnológico de Costa Rica
Apdo. 159-7050, Cartago
Tel: (506) 2550-2297 / 2550-2336 / 2550-2392
Fax: (506) 2552-5354
Hecho el depósito de ley.
Impreso en Costa Rica.

*Para quienes son motivo de admiración y
agradecimiento hacia Dios que les permitió ver
la luz de un mundo que requiere de todos
nosotros para que sea cada día mejor.*

*A mi esposa Jetty y a mis hijas: Erika, Verónica
y Astrid.*



CONTENIDO

Prefacio	13
1 Introducción	17
1.1 Presentación histórica	19
1.2 Sistema de bases de datos versus sistema de archivos	30
1.3 Conceptos básicos	33
1.4 Arquitectura ANSI/SPARC	41
1.5 Logros de los sistemas de bases de datos	43
1.6 Arquitectura Cliente/Servidor	44
1.7 Tendencias actuales	48
Ejercicios y preguntas de repaso	50
2 Modelo de Datos Semántico	53
2.1 Introducción	55
2.2 El modelo de datos	58
Ejercicios y preguntas de repaso	73

3 Modelo Relacional de Codd

3.1 Introducción	79
3.2 Estructura del modelo	79
3.3 Reglas de integridad del modelo	87
3.4 Algebra relacional	89
3.5 Cálculo de predicados	107
3.6. Los 12 mandamientos de Codd	117
Ejercicios y preguntas de repaso	122

4 Lenguaje de Consultas SQL

4.1 Introducción	127
4.2 Configuración y entorno del SQL	128
4.3 Descripción de la base de datos	130
4.4 Definición de la base de datos	131
4.5 Manipulación de la base de datos	135
4.6 Otras cláusulas del SQL	143
4.7 SQL incorporado	150
4.8 Reglas de integridad	152
4.9 Limitaciones del SQL	153
Ejercicios y preguntas de repaso	156

5 Proceso de Normalización

5.1 Introducción	161
5.2 Primera forma normal (1FN)	164
5.3 Dependencias funcionales	166
5.4 Segunda forma normal (2FN)	177
5.5 Tercera forma normal (3FN)	179
5.6 Tercera forma normal de Boyce-Codd (3FNBC)	181

5.7 Dependencias multivaluadas y la cuarta forma normal (4FN)

5.8 Quinta forma normal (5FN)	186
Ejercicios y preguntas de repaso	188

6 Modelaje y Diseño de Bases de Datos

6.1 Introducción	195
6.2 Clasificación de metodologías	195
6.3 Diccionario de datos	199
6.4 Metodología de diseño	201
6.5 Transformación al modelo relacional	214
Ejercicios y preguntas de repaso	223

7 Integración de Esquemas de Aplicación

7.1 Introducción	229
7.2 Conflictos de integración	230
7.3 Metodología de integración de esquemas de aplicación	233
Ejercicios y preguntas de repaso	245

8 Evaluación y Optimización de Consultas

8.1 Introducción	249
8.2 Establecimiento del problema	251
8.3 Implementación de los operadores relacionales	254
8.4 Implementación del operador Θ-join	256
8.5 Optimizador de consultas	260
8.6 Descomposición de consultas	270
Ejercicios y preguntas de repaso	279

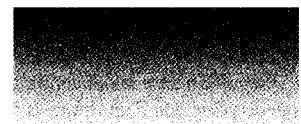
9 Control de Conurrencia

9.1 Introducción	283
------------------	-----

9.2	Transacción	284
9.3	Conflictos entre transacciones concurrentes	287
9.4	Itinerario	291
9.5	Mecanismos para controlar la concurrencia	294
9.6	Transacciones complejas y de larga duración	307
	Ejercicios y preguntas de repaso	310
10	Mecanismos de Seguridad e Integridad	313
10.1	Introducción	315
10.2	Control del acceso a los datos	316
10.3	La seguridad multinivel	322
10.4	Control de la inferencia	325
10.5	La criptografía	327
10.6	Reglas de integridad	331
	Ejercicios y preguntas de repaso	332
11	Recuperación después de fallas	335
11.1	Introducción	337
11.2	Transacciones concurrentes	338
11.3	Técnica de paginación	339
11.4	Técnica de la bitácora	345
	Ejercicios y preguntas de repaso	349
12	Bases de Datos Distribuidas	351
12.1	Introducción	353
12.2	Redes de computadores	353
12.3	Componentes básicos de un SABDD	356
12.4	Modelaje de bases de datos distribuidas	358
12.5	Mecanismos utilizados en un SBDD	362

12.6	Bases de datos distribuidas heterogéneas	374
12.7	Los nuevos retos del World Wide Web	380
	Ejercicios y preguntas de repaso	382
13	Máquinas de Bases de Datos	383
13.1	Introducción	385
13.2	Ventajas del enfoque máquina de bases de datos	386
13.3	Arquitectura de una máquina de bases de datos	387
13.4	Grado de paralelismo de las máquinas de bases de datos	390
13.5	Productos comercializados	392
	Ejercicios y preguntas de repaso	397
14	La Orientación a Objetos en Bases de Datos	399
14.1	Introducción	401
14.2	Paradigma de orientación a objetos (OO)	402
14.3	SABDs orientados a objetos y relacionales ampliados	407
14.4	SABD-OO	409
14.5	SABDs relacionales ampliados	422
14.6	Metodologías de análisis y diseño orientadas a objetos	428
	Ejercicios y preguntas de repaso	436
15	Otras Tendencias	439
15.1	Introducción	441
15.2	Bases de datos multimedia	443
15.3	Bases de datos deductivas	446
15.4	Bases de datos difusas	451
15.5	Bases de datos temporales	456

Bibliografía	461
Índice temático	481
Índice de autores y productos	489



PREFACIO

La tecnología de bases de datos relacionales es una de las herramientas más importantes con que se cuenta actualmente para la administración, ~~elaboración~~, automatizada, de grandes volúmenes de datos.

El presente libro tiene como objetivo fundamental servir de texto en un curso de bases de datos a nivel de Bachillerato o de Maestría en Computación o Informática. También, puede utilizarse como documento de consulta para todas aquellas personas involucradas en dicha tecnología.

El libro consta de 15 capítulos en donde se tratan los diversos temas del mundo de las bases de datos.

Se inicia con un capítulo introductorio en donde se hace un recorrido histórico de la tecnología de bases de datos y se definen los conceptos fundamentales de base de datos y de Sistema Administrador de Bases de Datos (SABD). Luego, se identifican las ventajas de un SABD con respecto a un sistema de archivos convencional. Además, se establecen los conceptos básicos en este campo, así como la arquitectura estándar de un SABD, sus funciones y los mecanismos que las respaldan. También, se hace una introducción a la arquitectura Cliente/Servidor. Finalmente, se concluye con un vistazo a las tendencias actuales en el

campo de los sistemas de bases de datos en donde se explican los términos básicos de estas tendencias.

Seguidamente, en el capítulo 2, se introduce un modelo de datos que puede verse como una síntesis de varios modelos de datos, entre otros, el modelo de datos de Entidad-Asociación de P. Chen [CHEN76], el Análisis Orientado a Objetos de Coad y Yourdon [COAD91] y la metodología de diseño de bases de datos de Fleming y von Halle [FLEM89]. Solo se introduce el modelo de datos en cuestión, sin establecer la conexión con el modelo relacional. Esta relación será estudiada en el capítulo 6 consagrado al diseño de bases de datos.

En el tercer capítulo, se estudia a fondo el modelo relacional, esto es, sus estructuras, operaciones, así como las reglas de integridad que se pueden definir. También, se hace un estudio de los lenguajes relacionales de consultas: algebraicos y predicativos. Finalmente, se concluye con una presentación de las reglas conocidas como los mandamientos del modelo relacional y que fueron establecidas precisamente por E. Codd, padre de dicho modelo.

El capítulo 4 se dedica al estudio del lenguaje relacional SQL. Si bien es cierto, varios autores lo critican y lo consideran como una base inapropiada para futuros desarrollos de los sistemas relacionales -multimedios, objetos complejos, etc.-, su estudio se justifica pues es en la actualidad un estándar y la mayoría de los SABDs relacionales cuentan con el SQL como lenguaje de consultas.

En el capítulo 5, se estudia una herramienta muy útil en el proceso de diseño de bases de datos relacionales. Esta herramienta, conocida como normalización, se establece como una serie de etapas, llamadas formas normales. El objetivo es aplicar dichas etapas a un conjunto de relaciones con anomalías de almacenamiento, y así obtener un conjunto de relaciones equivalentes, sin este tipo de problemas.

El capítulo 6 está dedicado a la presentación de una metodología de diseño de bases de datos, basada en el modelo semántico del capítulo 2. El objetivo de la metodología es construir un esquema conceptual de una

función administrativa dada, la que se denomina esquema de aplicación. Previamente, se hará una introducción del concepto de diccionario de datos y de su importancia en el proceso de diseño de cualquier base de datos.

En el capítulo 7 se introduce una guía complementaria que permite la integración de los esquemas de aplicación, con el fin de obtener el esquema conceptual de un sistema de información. Debido a la complejidad de los sistemas, éstos deben ser estudiados en forma parcial, de ahí la necesidad de contar con una metodología de integración de aplicaciones.

En el capítulo 8 se estudian los diferentes mecanismos que un SABD relacional debe brindar para responder en forma eficiente a las consultas de los usuarios y se establecen las bases de un optimizador de consultas.

En el capítulo 9 se introduce uno de los temas de mayor importancia en un ambiente de bases de datos, como es el control que debe brindar un SABD cuando varias transacciones de usuario se realizan en forma simultánea. Una vez que se define el término de transacción, se estudia el concepto de itinerario y se valoran cuáles de éstos serán consistentes. Se concluye el capítulo con una sección dedicada a las transacciones complejas y de larga duración.

El capítulo 10 se dedica al estudio de las diferentes técnicas que pueden garantizar la seguridad de un sistema: el control en el acceso de los datos, los controles de flujos, los controles de inferencia y la criptografía. Además, se estudian las diferentes posibilidades que, en materia de integridad, brindan los lenguajes relacionales.

En el capítulo 11, se introduce al lector en las técnicas para la recuperación de una base de datos, después de que algún tipo de falla se produzca. Además, se estudian los conceptos de punto de control y de diario o bitácora de transacciones.

En el capítulo 12 se consideran los conceptos fundamentales del ambiente de bases de datos distribuidas. Asimismo, se estudian las características principales que deben satisfacer los SABDs distribuidos y se estudian varios productos. Además, se establece la diferencia entre

bases de datos distribuidas homogéneas y bases de datos distribuidas heterogéneas.

En el capítulo 13 se examinan las llamadas máquinas de bases de datos y se analizan las ventajas de utilizar tales sistemas paralelos cuando se trabaja con bases de datos muy voluminosas y complejas.

En el capítulo 14, se profundiza en el tema de la orientación a objetos aplicado a los sistemas de bases de datos, así como la presentación de varios productos comercializados.

El capítulo 15 se dedica a las nuevas tendencias en investigación en el ambiente de bases de datos, como son la incorporación de nociones tales como el tiempo, el espacio y la incertidumbre en la información.

Agradecimientos

Deseo hacer patente mi agradecimiento a mis colegas del Departamento de Computación del Instituto Tecnológico de Costa Rica, quienes con sus observaciones, fueron motor importante para la mejora sustancial del presente material. En especial deseo agradecer a los compañeros: José Enrique Araya, Carlos Loría e Ignacio Trejos.

Asimismo, deseo agradecer a mis estudiantes de los cursos de bases de datos de los Programas de Bachillerato y Maestría en Computación, quienes fueron parte importante en la revisión de este material durante la impartición de dichos cursos.

Finalmente, deseo manifestar mi gratitud a la señora Paulina Retana por la edición y la revisión del texto. Sus sugerencias fueron muy atinadas y de gran valor.

A todos muchas gracias.
Carlos Alberto González A.



1

Introducción

«Condición esencial en un trabajo es no creer que la primera cosa que uno encuentra es la verdad. Uno puede estar errado y los otros que discuten también pueden estar errados. Hay que desconfiar de la ley. Sentir la obligación de renovar, de investigar, en condiciones que no sean de interés o egoísmo. No desconfianza que se convierta en inercia sino desconfianza que impulse a la búsqueda. La verdad es temporal. Lo que hoy parece cierto puede ser cambiado mañana a la luz de descubrimientos nuevos. Pero para esto se necesita paciencia.»

Clodomiro Picado T. (1887-1944),
biólogo costarricense.

1.1 PRESENTACION HISTORICA

Desde que el hombre tuvo uso de razón y se hizo sedentario, ha sentido la necesidad de guardar registros sobre sus diferentes actividades. Así, por ejemplo, en Mesopotamia, los antiguos comerciantes asirios, medos y persas, registraban datos haciendo incrustaciones en forma de cuñas sobre tabletas de arcilla. Este tipo primitivo de lenguaje escrito se conoce como *escritura cuneiforme*.

Por su parte, la civilización Inca, antes de que llegaran los europeos a América, guardaban datos por medio de cuerdas de colores en las cuales hacían nudos, y que llamaron *quipus*.

En tiempos más recientes, surge la era de la computación y en 1937 se construye el primer prototípico de computador electrónico, por John Vincent Atanasoff.

A partir de este momento, surge la imperiosa necesidad de almacenar, de forma permanente, la gran cantidad de datos que podía generar esta nueva herramienta. Así, se desarrollan las primeras técnicas de almacenamiento de datos con el fin de que pudieran ser posteriormente utilizados por el computador en forma precisa y rápida.

En 1964 es la primera vez que se menciona en la literatura el concepto de *base de datos*. En efecto, un Congreso que se llevó a cabo en la ciudad de Santa Mónica, California, se intituló “*Development and management of a computer-centered database*” [MIRA86].

Sin embargo, es importante recalcar que tales conceptos fueron anticipados por dos décadas. En efecto, en 1945, el profesor en Ingeniería Vannevar Bush, del Instituto Tecnológico de Massachusetts, publicó un artículo sobre el manejo y administración de microfilms, en el cual proponía las bases fundamentales de lo que luego se dio en llamar *tecnología de bases de datos* [WIED83].

En forma intuitiva, una *base de datos* -del inglés *database*- se define como un conjunto de datos almacenados en un dispositivo de almacenamiento masivo -como pueden ser discos duros, CD-ROM, etc.-,

el cual se encuentra disponible, en forma simultánea a un número de usuarios autorizados y en un tiempo pertinente.

Por su parte, el software que facilita la comunicación de los usuarios con la base de datos, por medio de un lenguaje de consultas y en donde se garantiza la integridad y la seguridad de los datos, así como la recuperación de la base de datos en caso de fallas, se llama *Sistema Administrador de Bases de Datos (SABD)* -del inglés *Database Management System (DBMS)*-.

Por su parte, al sistema que soporta una base de datos, es decir, el SABD junto con la base de datos se conoce como *Sistema de Bases de Datos*. A pesar de que la tecnología de bases de datos es relativamente joven, hasta la fecha, se puede hablar de tres grandes generaciones de bases de datos, las cuales se estudiarán a continuación.

1.1.1 Epoca anterior a las bases de datos

El primer gran intento por lograr el almacenamiento masivo de datos en un computador fue gracias a la aparición de la cinta magnética a mediados de los años cuarenta. Con este nuevo dispositivo, se podían guardar grandes cantidades de datos a costo razonable. Sin embargo, la noción de archivo, que se remonta a esta época, se reducía solo al denominado concepto de *archivo secuencial*, es decir, un archivo en que el acceso a los registros se hace en la misma forma en que los registros fueron almacenados, es decir, uno después del otro, sin dejar espacios.

En esta época la computación no había salido aun de los centros de investigación y aulas universitarias. La utilización de la computación en la administración se produce posteriormente y gracias a varios acontecimientos.

En primer lugar, los costos de almacenamiento empezaron a descender regularmente, mientras que la capacidad de procesamiento iba cada día en aumento.

En segundo lugar, se puede mencionar la invención de los *discos magnéticos* o discos duros. Es con este dispositivo que realmente se

pudieron desarrollar los primeros sistemas de archivos, así como sus diferentes tipos de organización -archivos directos, archivos indizados, etc.- pues permitían el acceso directo de los datos.

Entre los primeros sistemas de archivos se puede mencionar a *ISAM (indexed sequential access method)* y *VSAM (virtual sequential access method)*.

Otro hecho importante que motivó que la industria y el comercio utilizaran la computación como herramienta administrativa, fue la introducción en el mercado, en 1964, de la familia Sistema/360 de IBM.

Con este nuevo equipo y otros que surgieron en esta época, las empresas empezaron a tomar conciencia de la importancia y el poder que la información podía darles en el mejoramiento de la productividad, en el logro de mayor eficiencia, así como en la toma de decisiones.

Las empresas se hicieron cada vez más exigentes. Sin embargo, las restricciones de tipo técnico de los sistemas de archivos, no permitían responder satisfactoriamente a sus necesidades.

Por esta razón, varios grupos de investigación se dan a la tarea de buscar nuevas opciones de almacenamiento y administración de datos, que pudieran responder a los nuevos requerimientos empresariales.

1.1.2 Primera generación de bases de datos

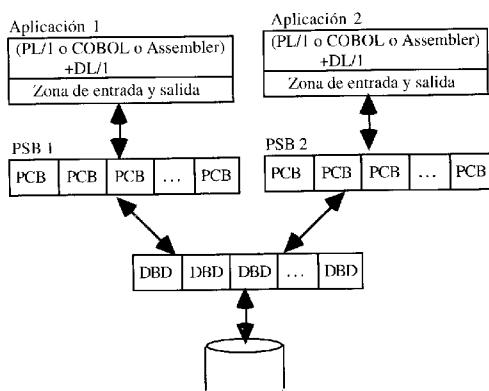
El origen del primer SABD se ubica a mediados de los años sesenta cuando las compañías IBM y Rockwell International desarrollaron las primeras versiones de un sistema conocido como Data Language/I (DL/I) con el propósito de administrar la gran cantidad de datos asociados con el proyecto espacial Apolo. Este sistema permitía el acceso a estructuras de datos jerárquicas desde programas en COBOL los cuales corrían bajo Sistemas/360 de IBM. Posteriormente IBM le adiciona un componente conocido como Information Control System/Data Language/I (ICS/DL/I), con el fin de permitir el acceso a los datos en forma concurrente.

Este primer producto se comercializa en 1969 bajo el nombre de *Information Management System 360 (IMS/360)* para los Sistemas/360.

En 1971 IMS/360 evoluciona al sistema IMS/360 Versión 2 y dos años después aparece el IMS/VS para los equipos 370 de IBM.

El IMS es un SABD de tipo jerárquico, es decir, se sustenta sobre un modelo de datos -estructura formal que permite la representación del mundo real- jerárquico o de árbol, en donde los nodos del árbol representan las entidades o conjuntos de objetos del mundo real, y los arcos entre los nodos, representan las posibles correspondencias entre dichas entidades. Este concepto será profundizado en el capítulo 2.

La arquitectura del SABD IMS se muestra en la figura 1.1.



DBD: Descripción de la base de datos
 PCB: Bloque de comunicación de programa
 PSB: Bloque de especificación de programa
 DL/I: Data Language/I

Figura 1.1 Arquitectura del SABD IMS

En primer lugar se tiene el *DBD* -Data Base Descriptions- o descripción de la base de datos el cual define una base de datos física y describe todos los segmentos de la base de datos -longitud, llave, etc.-.

Por su parte, el *PCB* -Program Communication Block- o bloque de comunicación de programa define una base de datos lógica y la correspondencia con la base de datos física.

Finalmente, el *PSB* -Program Specification Block- o bloque de especificación de programa define el conjunto de PCBs para un usuario dado.

A continuación se presenta un ejemplo de una base de datos sustentado en un modelo del tipo IMS.

Ejemplo. Considerar las entidades involucradas en un sistema de matrícula de los estudiantes de una universidad. En este caso se pueden establecer cuatro conjuntos de objetos o entidades:

- PROFESOR
- DEPARTAMENTO
- ESTUDIANTE
- CURSO

Además, cada una de estas entidades tienen atributos que las caracterizan. Por ejemplo, cada objeto estudiante se identifica por su carnet, su nombre y la dirección. Con respecto a las asociaciones, éstas son del tipo *uno a muchos*. En efecto, por ejemplo, para cada elemento profesor, existen varios estudiantes asociados y para cada estudiante se asocian varios cursos en los cuales se encuentra matriculado. En la figura 1.2 se representa este esquema -descripción lógica- de la base de datos bajo un enfoque jerárquico.

Los sistemas jerárquicos se caracterizan por ser muy rápidos, equivalentes a un sistema de archivos de acceso directo; sin embargo, en ellos no se tiene una división clara entre lo que es el nivel lógico y el nivel físico de la base de datos. Esto produce mucha rigidez a la hora de hacer cualquier modificación en la base de datos. En efecto, si se desea

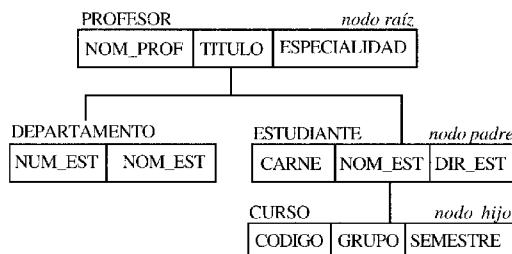


Figura 1.2 Ejemplo de una base de datos del tipo IMS

cambiar la estructura de la misma, por ejemplo, alterar la constitución de una entidad o variar la organización de almacenamiento, estos cambios requieren de un gran esfuerzo de reprogramación.

Entre las limitaciones de los sistemas jerárquicos, se pueden mencionar las siguientes:

- Debido a la propia definición del modelo de datos jerárquico, existen problemas para expresar asociaciones del tipo muchos a muchos, es decir, asociar un elemento de una entidad A con varios elementos de una entidad B y viceversa. Lo mismo se da con las asociaciones muchos a uno. Estos conceptos se profundizarán en el siguiente capítulo.
- Si se hacen modificaciones en el uso de la base de datos puede afectar sensiblemente el rendimiento del sistema.
- El costo de las consultas que no habían sido previstas en el diseño inicial de la base de datos puede ser muy alto.
- Los lenguajes de manipulación son procedimentales.
- Las operaciones de inserción y supresión de elementos se vuelven muy complejas, debido a un ordenamiento jerárquico muy rígido.
- Se puede tener mucho desperdicio de espacio al incurrir muchas veces, en redundancia de los datos.

- La supresión de un elemento de una entidad padre debe implicar la supresión de los hijos asociados. Si no se tiene cuidado, al aplicar esta operación, pueden quedar muchos elementos no asociados.

Otro sistema predominante en esta *primera generación* de bases de datos es el *System 2000* desarrollado por MRI Systems. La versión original fue diseñada para computadores CDC 6000. Las versiones posteriores se implementaron para sistemas 360 y 370 de IBM. El System 2000 es un SABD jerárquico que se dice *invertido* -o *indizado*- pues se sustenta en una organización de índice, el cual contiene algunos valores de campos, seguidos de la lista de apuntadores a los registros caracterizados precisamente por esos valores de campos. Este tipo de organización puede acelerar el acceso a los datos en forma considerable. Otros sistemas de este tipo son ADABAS de Software AG y DATACOM/DB de Applied Data Research.

1.1.3 Segunda generación de bases de datos

En forma paralela al desarrollo de los sistemas jerárquicos, aparecen los llamados SABDs *de redes*, los cuales representan un pequeño avance con respecto a los SABDs jerárquicos.

En efecto, con un sistema de redes, el mundo real se representa por medio de un grafo o red, en el que las entidades se conectan por medio de punteros lógicos.

Al igual que el modelo jerárquico, las asociaciones son del tipo uno a muchos. Sin embargo, permite una representación simétrica de las asociaciones muchos a muchos.

La utilización del modelo de red para el desarrollo de un SABD, se inicia con el sistema IDS -Integrated Data Store-, desarrollado por C.W. Bachman de la compañía General Electric. Sin embargo, fue propuesto definitivamente por el grupo DBTG -Data Base Task Group- del comité CODASYL -CONFERENCE ON DATA SYSTEMS LANGUAGES- en el año de 1969.

El comité CODASYL se establece en 1959 al reunirse los representantes de unas 50 organizaciones privadas y estatales con el fin de desarrollar y recomendar estándares en el desarrollo e implantación de sistemas computacionales. El primer gran proyecto de este comité fue el desarrollo de un lenguaje de programación orientado a la administración, llamado COmmon Business Oriented Language y conocido posteriormente como COBOL.

Los SABDs de redes, al igual que los jerárquicos, permiten compartir datos centralizados. Sin embargo, la dependencia de los datos aún persiste y se tiene un acceso navegacional muy tedioso. Además, los lenguajes de descripción y manipulación de datos son independientes y éstos últimos son procedimentales.

En la figura 1.3 se presenta la versión de redes del ejemplo de la figura 1.2.

Entre los SABDs que marcaron esta *segunda generación*, se pueden citar a DMS 1100 de UNIVAC, DMS-II de Burroughs, IDMS de Cullinet y TOTAL de la compañía Cincom System Inc.

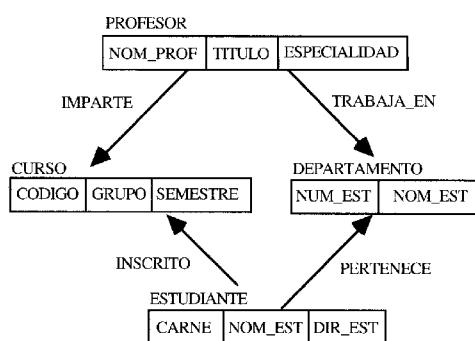


Figura 1.3 Ejemplo de una base de datos de redes

Entre los inconvenientes de los sistemas de redes, se tienen los siguientes:

- Son sistemas muy complejos pues los programadores de aplicaciones deben estar familiarizados con la estructura lógica de la base de datos, debido a la navegación que tiene que darse a través de todos los ítems.
- La representación de las asociaciones del tipo muchos a muchos no es natural y se debe usar un artificio. En efecto, si se tiene una asociación tal entre las entidades A y B, se deben construir dos asociaciones de uno a muchos y la siguiente entidad $A \times B = \{(a,b) / a \in A \text{ y } b \in B\}$ se relacionan en la asociación original)
- Al reorganizar la base de datos, se puede perder la independencia de los datos, sobre todo cuando se eliminan las entidades.

1.1.4 Tercera generación de bases de datos

La tercera generación de bases de datos se inicia con los trabajos del matemático inglés Edward Codd, con la propuesta de un nuevo modelo de datos. Este modelo, conocido como *modelo relacional de Codd*, en donde los datos se presentan en forma tabular, permitió acercar considerablemente el usuario final al desarrollador de la base de datos, situación que no se daba con los SABDs jerárquicos y de redes.

Esta nueva propuesta se da en 1970, en donde E. Codd sintetiza, en su famoso artículo en la revista Communications of the ACM: “*A relational model of data for large shared data banks*” [Codd70], las bases fundamentales de un modelo de datos en que las entidades y las asociaciones entre entidades iban a representarse por medio de relaciones, término sustentado en los conceptos matemáticos de *relación* y *el cálculo de predicados*.

Sin embargo, ya desde 1969, Codd había escrito un reporte técnico para IBM, con estas ideas [Codd69]. De esta forma, con los trabajos pioneros de Codd, se inicia la *tercera generación* de bases de datos, la cual da origen a los llamados *SABDs relacionales*.

Inicialmente, varios prototipos fueron desarrollados con el fin de establecer la factibilidad de implementar lenguajes de alto nivel no procedimentales y basados en el álgebra relacional o el cálculo de predicados. Entre estos, se pueden citar IS/1, XRM de IBM Cambridge Scientific Center, GMIS -Generalized Management Information System- del MIT Sloan School Energy Laboratory y RDMS de GMR [ASTR76], [KIM 79].

Ya para mediados de la década de los setentas, existían al menos, dos grandes prototipos relacionales.

En primer lugar, entre 1974 y 1978, se trabajó en el proyecto llamado *System R* desarrollado por 15 investigadores de IBM Research Laboratory en San José, California, bajo la dirección de W. Frank King [STON94].

En segundo lugar, se puede mencionar el proyecto *Ingres -Interactive Graphics and Retrieval System-* de la Universidad de California en Berkeley, coordinado por M. Stonebraker y Eugene Wong. Este proyecto que comienza en marzo de 1973, fue inicialmente implementado con ayuda del lenguaje de programación C y se utilizó un computador PDP 11/40 de Digital Equipment Corporation.

En la actualidad, el número de productos de bases de datos relacionales comercializados es superior a los 200. Entre los SABDs relacionales más importantes, tanto por el número de licencias vendidas, como por sus capacidades, se pueden citar entre otros, los siguientes:

- *DB2* de IBM,
- *Oracle* de Oracle Corporation,
- *Ingres* de Relational Technology,
- *SQL-Server* de Sybase,
- *Informix* de Informix Software Inc. y
- *SQL-Base* de Gupta Corporation.

Es importante mencionar que, al inicio de la era relacional se presentaron varios obstáculos y se hicieron muchas críticas a este nuevo enfoque. En efecto, si los SABDs jerárquicos y de redes conservaron aproximadamente el mismo tiempo de respuesta promedio que los archivos de acceso directo, los SABDs relacionales conocieron al inicio grandes problemas, sobre todo en lo relativo al tiempo de respuesta de las consultas.

Sin embargo, los problemas de rendimiento en los SABDs relacionales han sido superados; esto, debido a varios hechos.

En primer lugar, gracias a un hardware cada vez más poderoso -mayor rapidez de procesamiento, así como mayor capacidad de almacenamiento- y a un costo cada vez menor, los SABDs relacionales se han convertido en sistemas poderosos, con tiempos de respuesta aceptables. Por ello se han logrado ubicar en el primer lugar de ventas en ámbito mundial, en lo que concierne a software para administración y recuperación de datos.

En segundo lugar, se encuentra la incorporación a los SABDs de un programa denominado *optimizador de consultas*. Este programa tiene por misión el análisis y descomposición de las consultas de los usuarios en una sucesión de llamados de acceso y almacenamiento de la base de datos para así establecer las mejores estrategias de acceso a los datos. En efecto, se debe mencionar que en el mundo relacional las consultas son *declarativas*. Esto significa que el programador de aplicaciones no tiene que realizar un nuevo programa de aplicación cada vez que se requiera una consulta que no fue considerada previamente y, además, que no tiene que *navegar* por la estructura de la base de datos. En este sentido, es el SABD quien se encarga de escoger el mejor camino hacia los datos. Este concepto será profundizado posteriormente en el capítulo 8.

En el medio aun persiste una gran cantidad de empresas que trabajan con sistemas tradicionales de archivos. Por esta razón, se considera importante describir en la siguiente sección las ventajas y desventajas

que se presentan al trabajar con un sistema de archivos o por el contrario, con un sistema de bases de datos.

1.2 SISTEMA DE BASES DE DATOS VERSUS SISTEMA DE ARCHIVOS

Para comprender mejor las ventajas de implementar un sistema de información por medio de una base de datos y no un sistema de archivos, se usará el siguiente ejemplo.

Considérese una entidad financiera, la cual se encuentra legalmente autorizada a captar dinero, por medio de cuentas de ahorros. Así, la entidad decide implementar un sistema de archivos, con el fin de automatizar los diferentes procesos propios de esta función de cuentas de ahorros. Entre los archivos de tal sistema, el más importante será el referente a los clientes el cual se denomina **CLIENTE_AHORROS**.

Possiblemente, cada registro contará con la cédula, el nombre y la dirección de un cliente, así como el monto del ahorro. Entonces,

[2-320-789, 'Juan Barrios', 'Atenas de Alajuela', 123.000]

podría ser un registro de dicho archivo. Por otra parte, supóngase que una nueva ley ha sido aprobada en el Congreso de la República con el fin de que estas entidades financieras puedan brindar también un servicio de cuentas corrientes. Para automatizar dicho sistema, se recurre de nuevo a crear un sistema de archivos. Aquí también, un archivo fundamental puede ser **CLIENTE_C.CORRIENTES**, en donde sus campos podrían ser: número de la cuenta, nombre del cliente, cantón, provincia, teléfono y saldo.

Bajo estas nuevas políticas, el señor Juan Barrios decide asimismo, abrir una cuenta corriente. Debido a esto, será necesario introducir otro registro, por ejemplo,

[234-95, 'Juan Barrios', 'Atenas', 'Alajuela', 4605656, 45.000].

Así, la entidad financiera envía mensualmente, un informe sobre el estado de la cuenta corriente a cada cliente.

Sin embargo, el señor Barrios en un momento dado, debe trasladarse a otro lugar y por lo tanto cambiar de domicilio, por ejemplo, Curridabat de San José. Entonces, pide que se actualice el registro en el sistema de cuentas corrientes.

Ahora bien, bajo un proceder como el anterior se pueden apreciar varias desventajas.

En primer lugar, se tiene redundancia de la información, pues el nombre y la dirección se almacenan más de una vez.

Además, se darían inconsistencias si se actualiza el registro en cuentas corrientes y no se hace en cuentas de ahorros, pues se tendrían, por ejemplo, dos direcciones diferentes para el mismo cliente.

Finalmente, si se desea hacer una consulta que no fue prevista anteriormente, se debe realizar un programa para tal fin, provocando atrasos y gastos indeseables.

Así, al utilizar un sistema de archivos, se tienen varios inconvenientes, entre otros:

- Se da una redundancia no controlada en los datos.
- Las modificaciones de los archivos pueden generar inconsistencias y errores indeseables.
- Cada nueva consulta requiere la construcción de un nuevo programa. Así, se da una pérdida de tiempo en la programación de cada nueva solicitud.
- El proceso de modificación de programas existentes es muy costoso.

En la figura 1.4, se esquematiza un enfoque tradicional de archivos.

Por su parte, en un ambiente de bases de datos, estos problemas se minimizan. En efecto, se contaría con solo un archivo de clientes que respondería a las necesidades de los sistemas de cuentas corrientes y de ahorros en forma oportuna y por supuesto, gracias a los lenguajes de

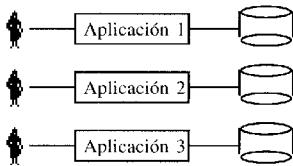


Figura 1.4 Enfoque tradicional de archivos

consultas de un SABD, cualquier nueva consulta que se deseé, será expresada en forma natural y es el sistema el que se encarga de hacer los procedimientos para responder a la solicitud planteada.

Entre las características más importantes que son comunes a las bases de datos se pueden citar las siguientes:

- La consulta a la base de datos se puede hacer en forma directa.
- Diferentes usuarios pueden tener acceso a la base de datos en forma simultánea.
- La redundancia de los datos se reduce.
- Con un buen diseño, se minimizan las inconsistencias de los datos.
- El proceso de modificación de programas existentes se reduce considerablemente.
- Se tienen controles centralizados para mantener la seguridad, privacidad e integridad de los datos

En la figura 1.5 se esquematiza el enfoque de bases de datos, en donde los datos se encuentran centralizados y es el SABD el encargado de buscar el camino óptimo para el acceso a estos datos.

A continuación se van a definir las características principales de un SABD, así como el personal involucrado en el ambiente de bases de datos.

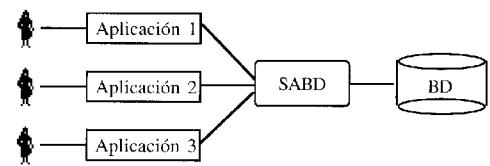


Figura 1.5 Enfoque de bases de datos

1.3 CONCEPTOS BASICOS

1.3.1 Lenguajes de consultas

Anteriormente, se mencionó cómo el SABD es el responsable del diálogo entre los usuarios y la base de datos. Para esto, el SABD dispone de un lenguaje de *definición de datos (LDD)* que permite la definición de las estructuras -entidades, registros, etc.- y de un lenguaje de *manipulación de datos (LMD)* que permite la inserción, modificación o supresión de los datos de la base de datos.

En el caso de los SABDs relacionales, estos dos lenguajes se encuentran integrados en uno solo que se llamará simplemente *lenguaje de consultas*. El lenguaje estándar ANSI/SQL es un ejemplo de este tipo de lenguaje el cual se estará estudiando en el capítulo 4.

Hasta la fecha se han propuesto más de 75 lenguajes de consultas relacionales, pero es el lenguaje SQL el que se ha convertido en un estándar.

Cuando un lenguaje de consultas es insuficiente para responder a los requerimientos de un usuario, es posible utilizar instrucciones de un lenguaje de consultas dentro de un lenguaje de programación de alto nivel convencional como C, COBOL y más recientemente C++.

Estos lenguajes, conocidos como *lenguajes huésped*, permiten el manejo de facilidades que no son propias del ambiente de bases de datos,

como por ejemplo instrucciones del tipo if-then-else, cómputos complejos, etc.

Cuando se hace un programa utilizando un lenguaje huésped, se conoce como *programa de aplicación*.

1.3.2 Funciones de un SABD

La función de un SABD no se limita únicamente a permitir, mediante la definición y manipulación de datos, el diálogo entre los usuarios y la base de datos.

Además, debe brindar mecanismos que permitan controlar la concurrencia de varios usuarios que deseen accesar la base de datos en forma simultánea; mantener la seguridad y la integridad de la base de datos; recuperar la base de datos y dejarla en un estado consistente, incluso después de que haya ocurrido una falla del sistema, ya sea ésta provocada por el hardware o por el software.

En la figura 1.6 se ejemplifican estas funciones.



Figura 1.6 Funciones de un SABD

Control de la concurrencia

Por la definición misma, una base de datos debe, en cualquier momento, estar disponible para que varios usuarios autorizados puedan accesarla en forma simultánea.

Para comprender mejor esto, considerar ¿qué ocurriría si, por ejemplo, dos usuarios se interesan simultáneamente en el mismo objeto -relación, tupla, etc.-?

En estos casos, llamados de *conflicto*, el SABD debe disponer de mecanismos que permitan arbitrar estas situaciones. Así, se implementan protocolos -conjuntos de reglas-, como por ejemplo el llamado *protocolo de dos fases*, que permiten que varias transacciones -ejecución de programas de aplicación- de usuarios se puedan realizar en forma concurrente, dejando la base de datos en un estado consistente, después de la intervención de los usuarios involucrados en la concurrencia.

En la figura 1.7, se muestra cómo, ante un conflicto -dos usuarios simultáneamente se interesan en el objeto A-, entonces el SABD debe hacer un análisis de la situación y según varios parámetros -importancia, edad de la solicitud, etc.-, tomar la decisión de asignar el recurso a uno de los usuarios involucrados.

El tema del control de la concurrencia y la resolución de conflictos, será estudiado con detalle en el capítulo 9.



Figura 1.7 Resolución de un conflicto

Seguridad

Supóngase que una organización cuenta con una base de datos que le permite organizar las diferentes funciones administrativas de la misma. En este caso, es natural pensar que no todos los datos deben estar disponibles a cualquier usuario. En efecto, si se piensa en una base de

datos que soporta un sistema de información integrado de una empresa, un usuario de aprovisionamiento no necesariamente debería tener acceso a la parte de la base de datos correspondiente a los salarios de los empleados.

Así, el SABD debe brindar mecanismos que permitan la confidencialidad de la información, por medio de controles de acceso, como "login" y "password". Esto permite que solo las personas que conozcan estas contraseñas puedan ingresar a la parte de la base de datos que se considera confidencial.

Existen otras técnicas que permiten garantizar la confidencialidad de los datos de una base de datos. Una de éstas es el encriptamiento, que consiste en traducir los datos en un criptograma por medio de un código. Así, solo el que posea tal código puede interpretar dichos datos.

Otras dos técnicas son el control del flujo de los datos por medio de clases de seguridad y el control de la inferencia de los datos que persigue que un usuario no pueda inferir datos privados a partir de datos a los que sí tiene acceso. Este tema se estudiará en el capítulo 10.

Integridad

Otro punto importante de tomar en consideración es el de la integridad de la base de datos.

Así, un SABD debe brindar la posibilidad de definir reglas de integridad que ayude a la verificación semántica y sintáctica de los datos. Por ejemplo, una regla de integridad podría ser, en una relación EMPLEADO, que el salario de los empleados siempre sea superior a 50.000 colones. Existen reglas de integridad de tipo entidad, de dominio y de referencia, que pueden ser implementadas con el lenguaje de consultas de algunas versiones recientes de productos relacionales.

Otras reglas, como las llamadas operaciones de disparo, requieren la programación de las mismas.

En la figura 1.8 se aprecia el caso de un usuario que desea introducir un objeto a la base de datos. La inserción de dicho objeto será

satisfactoria o no dependiendo si cumple o no la regla de integridad que se definió previamente.

El establecimiento de las reglas de integridad es una parte importante en todo desarrollo de bases de datos. Sin embargo, debe siempre buscarse un equilibrio en las reglas que se establecen, pues dependiendo del número de las mismas, el rendimiento del sistema se puede ver afectado.

En los capítulos 3 y 4 se profundizarán los conceptos de los diferentes tipos de las reglas de integridad.

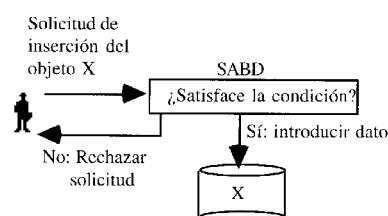


Figura 1.8 Regla de integridad en una base de datos

Recuperación luego de una falla

Supóngase que una transacción realiza la transferencia de fondos de una cuenta corriente a otra. ¿Qué pasa si en el preciso momento en que se ha hecho un débito de una cuenta A de 1.000 colones, antes de acreditar a la cuenta B, ocurre una falla? Esta situación se ilustra en la figura 1.9.

En este caso, el SABD debe garantizar que la base de datos no quede en un estado inconsistente. Por eso, debe manejar las transacciones como de ejecución atómica, es decir, se realiza toda la transacción o no se realiza del todo, borrando todo el trabajo que una transacción de usuario haya podido hacer si ésta aun no ha llegado a su punto de validación.

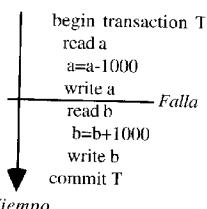


Figura 1.9 Ejemplo de falla en una transacción

El SABD al invocar a lo que se llama bitácora, permite, según sea el caso, hacer, rehacer o deshacer los cambios generados por una transacción de usuario. El problema de la recuperación después de una falla será profundizado en el capítulo 11.

1.3.3 Personal del ambiente base de datos

En los primeros SABDs, las personas que se involucraban en el ambiente de bases de datos estaban muy bien establecidas. Eran profesionales con gran conocimiento en áreas como organización de archivos, lenguajes de programación, sistemas operativos, etc. Así, se tenían fronteras bien definidas; por un lado, los responsables de la base de datos y, por el otro, los usuarios de ésta.

Hoy en día, con los sistemas relacionales, estas fronteras se han reducido considerablemente. Por ejemplo, con un sistema jerárquico era sumamente difícil que un usuario pudiera realizar una consulta personal a la base de datos. En cambio, con un sistema relacional el usuario puede efectuar consultas del tipo *ad-hoc*, utilizando un lenguaje de consultas como SQL.

Dependiendo de la complejidad de la base de datos que se deseé desarrollar, varias de las funciones previstas en este desarrollo pueden ser asumidas por una sola persona o bien, por varios grupos de personas.

Así, una base de datos puede ir desde un sencillo conjunto de relaciones para llevar asuntos de una persona en particular o podría alcanzar varios Terabytes -1 Terabyte = 1024 Gigabytes-, como la base de datos con que actualmente cuenta la compañía TRW Information Services, en California, EE.UU.

A continuación, se presentan las diferentes funciones que deben ser asumidas por el personal involucrado en el desarrollo, mantenimiento y uso de una base de datos.

El usuario final

El usuario final es la persona más importante de quienes tienen relación con una base de datos. Así, es él quien determina el éxito de la base de datos, según la utilización que haga de la misma y si realmente se adapta a sus necesidades. Por ello, en todo desarrollo de bases de datos la comunicación con el usuario debe ser permanente y fluida.

Si se considera una base de datos en una entidad financiera, los usuarios pueden ser los cajeros, encargados de la contabilidad, gerentes, etc.

Con respecto a los usuarios se pueden clasificar en tres categorías: los que se interesan solo en la lectura de la base de datos, los que pueden introducir y suprimir datos y aquellos que pueden modificarlos.

El analista de datos

Es el encargado de establecer la comunicación inicial con los usuarios finales, quienes ayudan a establecer los límites y las responsabilidades del dominio del problema que se está estudiando.

Utilizando una metodología de desarrollo de bases de datos, establece las entidades y asociaciones entre ellas, así como las reglas de integridad que regirán la consistencia de la futura base de datos.

El analista estratégico

Cuando el mundo que se desea modelar e implementar por medio de una base de datos es muy complejo, es recomendable descomponer el

problema en sub-problemas y posteriormente aplicar una herramienta que permita integrarlos y obtener así el sistema integral. Es en la coordinación de los diferentes sub-sistemas realizados por los diferentes grupos de analistas, que entra a jugar el analista estratégico, cuya función es la coordinación de estos diferentes grupos de trabajo con el fin de que el paso posterior de integración de sistemas se realice en forma satisfactoria.

El implementador de la base de datos

El implementador de la base de datos es el encargado de trasladar el esquema conceptual de la base de datos, -una visión lógica de la misma- en estructuras denominadas relaciones. Esto lo realiza utilizando la parte de la descripción del lenguaje de consultas del SABD. Asimismo es el responsable de la carga inicial de la base de datos. También, debe programar las diferentes reglas de integridad que se establecieron en el esquema conceptual.

El programador de aplicaciones

Es el encargado de escribir códigos -programas de aplicación- para procesar las consultas de los usuarios. Por ejemplo, en un sistema de información que administra las finanzas de una compañía, el pago de la nómina puede verse como un programa de aplicación. Con los sistemas relacionales, la línea entre el usuario y el programador de aplicaciones no está bien definida, gracias al uso de lenguajes no procedimentales.

El ABD

Una persona -o grupo de personas- que es de suma importancia para mantener la base de datos en un estado consistente y que su utilización sea lo más eficiente posible, es el *administrador de la base de datos (ABD)*.

El ABD es quien sirve de puente entre la base de datos y los usuarios finales. Debe ser una persona con un alto conocimiento de la tecnología de base de datos, así como del SABD que se está utilizando. Asimismo,

debido a un acelerado desarrollo de los sistemas de bases de datos en ambientes distribuidos, el ABD debe contar con conocimientos en redes de computadores. En síntesis, el ABD tiene a su cargo, entre otras, las siguientes funciones:

- Dar mantenimiento al diccionario de datos. El diccionario de datos, ya sea automatizado o manual, lleva un recuento sobre el significado de los objetos de la base de datos, las asociaciones entre ellos, así como las reglas de integridad.
- Establecer y afinar la estructura física de la base de datos.
- Realizar cambios en los métodos de acceso o en las estructuras de la base de datos, según las necesidades de los usuarios y el mejoramiento del rendimiento de la base de datos.
- Asignar códigos de acceso, con el fin de mantener la seguridad y confidencialidad de la base de datos.
- Implementar mecanismos de recuperación de la base de datos en caso de una falla.
- Si se trabaja en un ambiente distribuido, verificar el acceso desde diferentes sitios.

1.4 ARQUITECTURA ANSI/SPARC

Según lo que ha definido el grupo de Estudios sobre SABDs ANSI/SPARC, la arquitectura de un SABD se divide en tres niveles: el nivel interno con su esquema interno o físico, el nivel conceptual con su esquema conceptual, y el nivel externo con sus esquemas externos o subesquemas, según se aprecia en la figura 1.10.

1.4.1 Nivel interno

El nivel interno o físico se refiere a la forma en que los datos se encuentran almacenados físicamente -en discos y cintas-, las estructuras de datos utilizadas, mecanismos de acceso y cómo se reparten los datos en los bloques que conforman la base de datos.

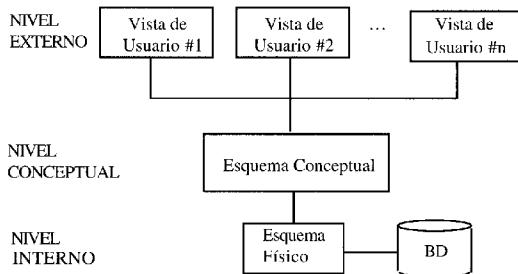


Figura 1.10 Arquitectura ANSI /SPARC de un sistema de bases de datos

El nivel interno es la representación de bajo nivel de la base de datos.

1.4.2 Nivel conceptual

El nivel conceptual, con su esquema conceptual, se compone del conjunto de datos lógicos que modelan el mundo real. Así, este mundo real será representado utilizando un modelo de datos -herramienta que permite modelarlo-. Por ejemplo, en el caso de un modelo de datos relacional, el esquema relacional se compone de un conjunto de estructuras llamadas relaciones.

Por su parte, gracias al lenguaje de consultas, se podrán definir estas relaciones en un formato que sea comprendido por el SABD.

1.4.3 Nivel externo

El nivel externo, por su parte, comprende un conjunto de vistas o subesquemas individuales de grupos de usuario -programadores de aplicaciones, usuarios finales-, el cual representa los datos lógicos específicos a una aplicación.

Cada vista se describe por medio de un esquema externo, que se define como la parte del esquema conceptual pertinente a la vista correspondiente.

1.4.4 Independencia física y lógica

Una vez establecida la arquitectura de un SABD, se pueden introducir los términos de independencia física e independencia lógica de los datos. En efecto, los SABDs relacionales gozan de la capacidad de variar la organización interna de los datos, por razones de rendimiento, sin afectar los programas de aplicación. Esto se conoce como *independencia física* de los datos. Asimismo, la posibilidad de modificar el esquema conceptual -relaciones, atributos- sin afectar los programas de aplicación, se conoce como *independencia lógica* de los datos.

Es importante señalar que esta independencia entre los diferentes niveles es muy débil en los sistemas jerárquicos y de redes, lo que los hace poco flexibles.

1.5 LOGROS DE LOS SISTEMAS DE BASES DE DATOS

En esta sección se discuten en forma resumida los grandes logros tecnológicos del modelo relacional en los últimos treinta años.

En las últimas tres décadas se pueden establecer tres grandes avances de los sistemas de bases de datos.

En primer lugar, el surgimiento y consolidación de los *sistemas de bases de datos relacionales*. Esto se debió a los siguientes hechos, que serán estudiados a través del libro:

- Desarrollo de lenguajes de consultas de alto nivel que permitieran un acceso sencillo a los datos.
- Desarrollo de la teoría y algoritmos para la optimización de consultas.
- Formulación de la teoría de la normalización.
- Desarrollo de algoritmos para la asignación de tuplas en páginas.
- Construcción de técnicas de indexación para brindar accesos rápidos.

En segundo lugar, se tiene la *administración de las transacciones*. En estos últimos años se establecen protocolos que permiten el acceso concurrente y eficiente de varios usuarios a los mismos recursos de la base de datos, así como las estampillas de tiempo. También, se establecen los mecanismos de recuperación después de una falla, como son el establecimiento de las bitácoras y las técnicas de archivos espejo.

Finalmente, otro gran logro ha sido el de las *bases de datos distribuidas*, esto es, una base de datos que se puede distribuir entre varios sitios y en la cual un usuario puede accesarla de manera tal que la distribución es transparente para él. Asimismo, se establecieron los protocolos de concurrencia para ambientes distribuidos así como la optimización de consultas.

1.6 ARQUITECTURA CLIENTE/SERVIDOR

En años recientes la llamada arquitectura *cliente/servidor* se está utilizando cada vez con mayor vigor, pues permite la comunicación de grandes volúmenes de datos dispersos en distintos sitios y en diferentes plataformas computacionales y bases de datos. Esta comunicación se hace mediante una red de computadores y se utilizan herramientas interactivas.

Lo que persigue este enfoque es el manejo de ambientes distribuidos cada vez más complejos y en donde se interconectan gran variedad de sistemas de computación.

En forma intuitiva, se puede decir que la arquitectura cliente/servidor establece, en un ambiente de computación distribuida, una relación entre dos componentes principales en la cual uno solicita requerimientos y el otro componente brinda los servicios para responder a esos requerimientos. Además, los papeles pueden llegar a invertirse, según sea la naturaleza de la solicitud y quién la haga.

Una arquitectura típica de cliente/servidor se puede apreciar en la figura 1.11, con sus tres componentes fundamentales:

- Uno o varios clientes
- Una red de computadores
- Uno o varios servidores

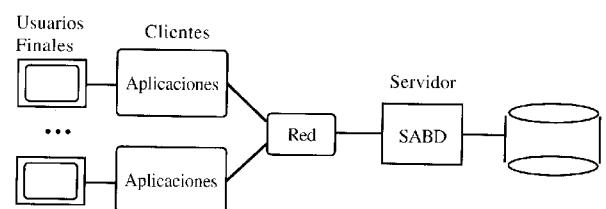


Figura 1.11 Arquitectura Cliente/Servidor en un sistema de bases de datos

Algunos autores afirman que hablar de cliente/servidor es hablar de *computación cooperativa*. En tales ambientes existe una relación entre varios componentes de software, los cuales interactúan con el fin de satisfacer los requerimientos de los usuarios.

1.6.1 Cliente

En el componente *cliente* se define el mundo en la forma en que lo percibe el usuario final y por ende, debe ser lo suficientemente versátil para que pueda responder a éste con los datos requeridos -gráficos, consultas, reportes, estadísticas, etc.-.

Es en este nivel en donde se establecen las aplicaciones, ya sean éstas desarrolladas por los usuarios, o bien aquellas suministradas por los proveedores de software, también llamadas *herramientas* -del inglés *tools*- y que pueden ser de varios tipos como generadores de aplicaciones, hojas de cálculo, escritores de reportes, paquetes estadísticos, etc. Las aplicaciones residen en los diferentes computadores de los usuarios y serán llamados asimismo *máquinas de los clientes*.

El envío de solicitudes y respuestas entre los clientes y el servidor se hace utilizando una interfaz de programa de aplicación, conocida como *API* -del inglés *application program interface*-.

Existen dos tipos de API. En primer lugar, se tienen las APIs incorporadas, en las cuales las instrucciones de tipo SQL se incorporan en el código de la aplicación. Por otra parte, se tienen las APIs a nivel de llamados, en donde el programador puede emitir instrucciones de SQL utilizando un lenguaje huésped.

Con el fin de establecer la comunicación entre clientes y servidores de bases de datos variados, sin tener que adaptar la aplicación a cada servidor específico, se está buscando la estandarización de dichas APIs.

1.6.2 Servidor

Cuando se habla de *servidor*, se hace referencia principalmente al SABD y es independiente del hardware en que se instala. Por lo anterior, debe cumplir con todas las funciones que caracterizan a un SABD relacional. Así, debe brindar facilidades que permitan la administración de los datos, y entre éstas, se deben tener al menos las siguientes [DALE90]:

- Un diccionario de datos,
- Memorias temporales y de caché,
- Mecanismos para satisfacer las solicitudes de los usuarios,
- Mecanismos de bloqueo y control de la concurrencia,
- Seguridad de datos,
- Mecanismos de recuperación y
- Mecanismos de configuración.

Es importante mencionar que, además de servidores de bases de datos, existen otros tipos, como por ejemplo, servidores de correo electrónico, servidores de documentación, servidores de comunicaciones, servidores de impresión, etc.

En la actualidad, existen varios productos relacionales que ofrecen servidores SQL, entre otros:

- Microsoft/Sybase SQL Server
- Oracle Server
- SQLBase Server de Gupta

En estos casos, cada cliente debe formular las consultas en SQL y ofrecer funciones de interfaz de usuario y de lenguajes de programación.

1.6.3 Red

En una arquitectura cliente/servidor, el componente de red no es otra cosa que un medio de transmisión de datos, donde los mensajes son intercambiados por medio de protocolos estándar de comunicación, como por ejemplo DecNet, TCP/IP, etc.

Al referirse al componente de red, se deben al menos considerar los siguientes aspectos [GOLD90]:

- Técnica de transmisión -tipo de banda-
- Tipo de cableado -cable coaxial, fibra óptica, etc.-
- Topología -bus, árbol, estrella, anillo-
- Protocolo de acceso -cómo compiten las máquinas para accesar la red-
- Protocolo de comunicación -TCP/IP, SPX/IPX, etc.-

Los formatos y protocolos de comunicación entre los clientes y servidores se denominan FAPs -del inglés *formats and protocols*-.

Al igual que los APIs, se está buscando la estandarización de los FAPs, específicamente el ISO Remote Database Acces -RDA-, que define FAPs basados en los protocolos de red OSI.

En la figura 1.12, se muestran los diferentes pasos que se deben seguir para el diálogo entre los clientes y el servidor.

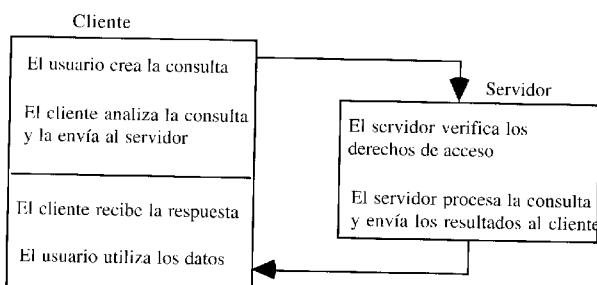


Figura 1.12 Pasos para tener acceso al servidor

1.7 TENDENCIAS ACTUALES

Los SABDs relacionales se han establecido como una tecnología consolidada. Sin embargo, muchas aplicaciones de multimedios que requieren el manejo de objetos complejos, como diseño y manufactura por computadora, necesitan otro tipo de administradores de datos.

Por ello, ya se han comenzado a investigar nuevos enfoques, que permitan llenar esta laguna e inclusive, se puede afirmar que se está entrando a la llamada *cuarta generación de SABDs*.

Los modelos de datos de estos sistemas se basan en el paradigma de *Orientación a Objetos (OO)*, que es en la actualidad uno de los más populares en el dominio de los lenguajes de programación.

La orientación a objetos se basa en varios conceptos como son *objeto, clase, tipo, herencia, encapsulación, sobrecarga*, etc. Todos estos conceptos serán ampliados en el capítulo 14.

Algunos proyectos se encuentran en marcha con el fin de estudiar los medios para fusionar las capacidades de un lenguaje orientado a objetos (OO) y las de un SABD relacional.

En lo que concierne a los SABD-OOs, éstos aun no han llegado a su madurez. Algunos usuarios de aplicaciones específicas, como CAD/CAM, ingeniería de software, telecomunicaciones o multimedios, ya los utilizan y han podido evaluar las posibilidades que encierra la orientación a objetos.

En lo que respecta a los nuevos tipos de aplicaciones, los productos relacionales actuales son inadecuados, no así el modelo relacional, pues muchas características del modelo relacional no fueron retenidas por los diferentes fabricantes de SABDs relacionales a la hora de desarrollar sus sistemas. Debido a las limitaciones de los SABDs actuales en estos momentos se pueden distinguir dos grandes tendencias: *ampliar los SABDs relacionales* o bien *desarrollar nuevos SABD-OOs*.

Otra tendencia importante es la referente al problema de la administración inteligente de la información. Esto es, módulos que, junto con SABDs evolucionados, permitan una recuperación fácil y rápida de la información requerida por el usuario. Esto constituye la *quinta generación de SABDs*. Esta información no solo se encontrará almacenada en una base de datos, sino que se podrá generar nueva información a partir de estos módulos. De esta forma, se espera contar con lenguajes de consultas en forma natural que permitan diálogos intuitivos con el usuario, así como poderosas herramientas de adquisición y representación de conocimiento, en donde éste podrá ser difuso, es decir, que se permita el manejo de información incierta y vaga, tan presente en la mayoría de los fenómenos naturales o que son manejados por el hombre.

Estos sistemas, que se pueden denominar SABDs *deductivos* son motivo de grandes esfuerzos de investigación y se puede decir que los primeros sistemas deductivos harán su aparición a inicios del próximo milenio.

A pesar de que no existe consenso sobre lo que debe ser un SABD deductivo, al menos sí debe satisfacer las siguientes propiedades [TSUR91]:

- Permitir la expresión de las consultas por medio de reglas lógicas.
- Permitir consultas recursivas y algoritmos eficientes para su evaluación.
- Poseer propiedades avanzadas, como por ejemplo negaciones estratificadas.
- Incorporar métodos de optimización que garanticen la traducción de las especificaciones declarativas en planos de acceso eficientes.
- Permitir dominios para objetos complejos.

Para concluir con este capítulo, en la figura 1.13 se sintetizan las diferentes épocas que han marcado el desarrollo de la tecnología de bases de datos.

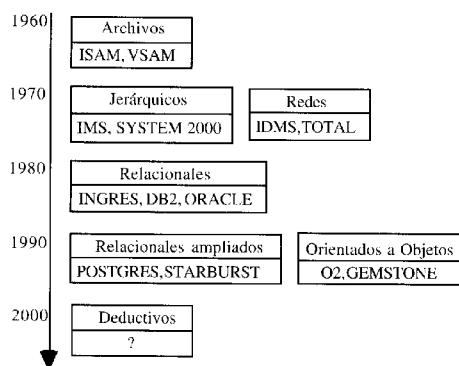


Figura 1.13 Las diferentes épocas de la tecnología de bases de datos

EJERCICIOS Y PREGUNTAS DE REPASO

1.1 Definir los siguientes conceptos:

- base de datos
 - IMS
 - DBTG
 - optimizador de consultas
 - DB2
 - lenguaje de definición de datos
 - lenguaje de manipulación de datos
 - lenguaje de consultas
 - lenguaje huésped
- 1.2 Establecer las características principales de un SABD relacional
 1.3 ¿Qué hechos motivaron el surgimiento de la tecnología de bases de datos?
 1.4 Explicar la arquitectura del SABD IMS.
 1.5 ¿Cuáles fueron los primeros prototipos relacionales?
 1.6 Dar una lista de 6 SABDs relacionales comercializados.
 1.7 Explicar cada una de las generaciones de la tecnología de bases de datos.
 1.8 ¿Cuáles son las ventajas del enfoque relacional con respecto de un sistema de archivos?
 1.9 Explicar la arquitectura ANSI de un sistema administrador de bases de datos relacional.
 1.10 ¿Qué es la independencia física y la independencia lógica de los datos?
 1.11 ¿Por qué se dice que un lenguaje de consultas relacional es no procedimental?
 1.12 En la arquitectura cliente/servidor, definir los siguientes conceptos:

- cliente
 - servidor
 - componentes principales del componente red
- 1.13 ¿Cuáles son las características principales de un SABD orientado a objetos?
- 1.14 Definir lo que significa un SABD relacional ampliado.
- 1.15 Explicar el concepto de base de datos deductiva.



2

Modelo de Datos Semántico

«La realidad y la apariencia se identifican mientras no se proyecten en la pizarra oscura de la nada, mientras no sean metafísicamente consideradas. Cuando la reflexión metafísica muestre la recíproca dependencia del ente y la nada, encontrará asidro, como antinomia de la creencia pura, la duda sobre si todo es plenitud de plenitudes o vanidad de vanidades. Pero esta opinión límite no tiene nada que ver con el método reductivo consistente en explicar lo inmediatamente dado, reputado apariencia pura, mediante un modelo que se supondría ser la pura realidad o la pura racionalidad. Lo infinito de la apariencia autosuficiente sólo encuentra su límite ante la nada.»

Roberto Murillo (1939-1994),
filósofo costarricense.

2.1 INTRODUCCION

Desde los albores de la humanidad, el ser humano ha sentido siempre una profunda necesidad de comprenderse a sí mismo y a su entorno. Esta necesidad de responder a preguntas clásicas como ¿por qué?, ¿cómo?, ¿de dónde vengo?, ¿hacia dónde voy?, etc., han dado origen a lo que hoy en día se entiende por *conocimiento*.

El conocimiento, entendido como el hecho de tener una idea sobre un objeto, nace de dos fuentes: la experiencia y el pensamiento.

Una vez que el hombre adquiere conocimiento sobre diferentes hechos, fenómenos, etc., siente la necesidad de transmitir ese conocimiento adquirido a otros seres humanos, generando así lo que se da en llamar *conocimiento colectivo*. Pero esta transmisión de conocimiento requiere de un medio. Nace entonces, el concepto de *dato*, palabra que proviene del latín *datum* y significa *hecho*.

Sin embargo, el conocimiento colectivo cada día es mayor y por lo tanto se generan grandes cantidades de datos. Esto hace necesario buscar herramientas que permitan su almacenamiento.

Con respecto del ser humano y su entorno, se puede hablar de tres grandes niveles de almacenamiento de datos.

En primer lugar, se tiene la *memoria genética*. En ésta, que se encuentra en el ADN de cada una de las células de un organismo viviente, se genera la información responsable de determinar las bases fisiológicas de la vida como el tipo de cabello, la acción de respirar, de amamantarse, etc.

En segundo lugar se puede hablar de la memoria localizada en el *cerebro*. En ella se almacenan datos que permiten incrementar el conocimiento.

Sin embargo, la memoria del cerebro es insuficiente para almacenar todo el conocimiento colectivo. Nace así, la *memoria masiva o externa* y es donde se almacenan grandes cantidades de datos. Entre estas

memorias masivas se tienen los libros así como las cintas y los discos de los sistemas computacionales.

Una de las herramientas más ampliamente utilizadas en la historia de la humanidad para trasmisir y almacenar datos ha sido el *lenguaje natural*. Este instrumento tiene la virtud de poder representar conjuntamente la interpretación de los datos -su significado-, así como los datos mismos [TSIC82].

A este significado o interpretación que se puede dar de los datos es lo que se conoce como *información*.

En cuanto al lenguaje natural, lo ideal sería utilizarlo en un sistema computacional para representar información, sin embargo, esta posibilidad aun es incierta, debido a que es muy general y ambiguo y se presta a múltiples significados. Así, en muchos casos es mejor contar con una herramienta especializada. Por ejemplo, la notación científica muestra la necesidad de contar con modelos especializados para la representación de la información. Además, el lenguaje natural es una herramienta que evoluciona libremente entre los humanos y como tal no es precisa para almacenar y transmitir información. Finalmente, posee propiedades que lo hacen muy difícil de manipular por medio de operaciones de un computador.

Por esta razón es de interés, en este contexto, construir herramientas que puedan representar información y que sean fácilmente utilizables por medio de un sistema computacional. Estas herramientas, de las cuales se introduce una en este capítulo, se conocen como *modelos de datos*.

Un *modelo de datos* es una herramienta que se compone, por una parte, de un formalismo que describa los datos y, por otra, de un conjunto de operaciones que permitan manipularlos. Además, se puede adicionar un tercer componente y es el referente al conjunto de reglas que mantienen la integridad de los datos, es decir, que permiten verificar cuáles datos son válidos dentro de un sistema y cuáles deben ser rechazados.

Como se vio en el primer capítulo, en los inicios de la tecnología de bases de datos, se utilizaron modelos de datos basados en las estructuras de jerarquías y redes y se pudieron establecer las limitaciones inherentes de estos modelos para representar información.

En este capítulo se estudiará un modelo de datos de tipo semántico, el cual es una ampliación del modelo Entidad-Asociación de Chen [CHEN76]. Se dice que es semántico porque su característica principal es brindar un mayor contenido semántico que el modelo de datos relacional.

Existe mucha literatura referente a los modelos de datos semánticos. Entre otros, se pueden citar los siguientes:

- *IFO* [ABIT87]
- *NIAM* (Nijssen's Information Analysis Methodology) [NIJS89]
- *SHM+* (Extended Semantic Hierarchy Model) [BROD84]
- *RM/T* (Tasmanian Model) [CODO79]

La presentación de un modelo de datos semántico en un libro dedicado a los sistemas de bases de datos relacionales se justifica ya que, en muchas metodologías de diseño de bases de datos relacionales, se utilizan los modelos de Entidad-Asociación o ampliaciones de éste, para hacer el análisis del dominio del problema considerado, y obtener así un esquema lógico, el cual posteriormente se traduce a un esquema lógico relacional equivalente.

En efecto, según se aprecia en la figura 2.1, para implementar un sistema de información, por medio de una base de datos relacional, se deben cumplir varias etapas.

En primer lugar, se debe discernir del mundo real, la parte que será pertinente a la aplicación que se desea modelar y que se denomina *dominio del problema*.

Una vez que se delimiten los alcances de la aplicación, ésta se modela utilizando el modelo de datos semántico, obteniendo un *esquema lógico*

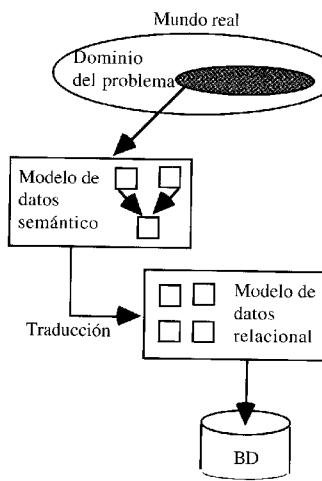


Figura 2.1 Proceso de modelaje de una aplicación

-también llamado conceptual-. Finalmente, se procede a la traducción de este esquema a uno equivalente relacional, es decir, que represente la misma información del dominio del problema.

2.2 EL MODELO DE DATOS

Un campo de investigación muy importante actualmente es el referente al modelaje de datos, entendido este concepto como el desarrollo de modelos y metodologías que permitan representar la información del mundo real en una forma cada vez más fiel.

El modelo de datos que aquí se presenta se fundamenta en cinco conceptos principales que permiten establecer la estructura y manejar la complejidad de los sistemas.

A pesar de que el modelo aquí presentado es simple, esto no disminuye su capacidad de representar fielmente muchas situaciones del mundo real que son complejas.

Los elementos principales del modelo son los siguientes:

- Entidad
- Asociación
- Atributo
- Herencia
- Composición

A continuación, se estudiará cada uno de estos conceptos.

2.2.1 Entidad

Una de las tres formas de organización que utiliza constantemente el ser humano para comprender el mundo real es la representación de éste en clases y elementos, así como la distinción que se puede hacer entre las diferentes clases de elementos [COAD91].

Así, lo que interesa, en primer lugar, es definir el concepto de objeto dentro del contexto del modelo de datos aquí introducido. En este caso, un *objeto* es cualquier cosa concreta o abstracta del mundo real.

En segundo lugar, interesa agrupar los objetos similares en conjuntos llamados *entidades*. Así, una entidad es un conjunto de objetos que son agrupados debido a que tienen características semejantes, según el contexto de la aplicación que se está modelando.

Ejemplo. Pedro, María, Juan, son objetos de la entidad PERSONA, según se aprecia en la figura 2.2.

Un hecho importante del modelo de datos es la representación gráfica de los distintos componentes que lo conforman. Así, para una entidad, su representación gráfica será un rectángulo con dos zonas, como se

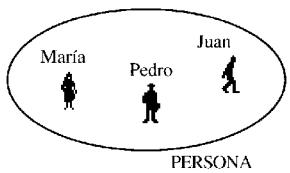


Figura 2.2 Ejemplo de una entidad

muestra en la figura 2.3. En la parte superior se colocarán los atributos que conforman la llave primaria de la entidad y el resto de los atributos se colocan en la parte inferior.

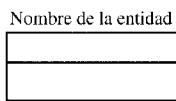


Figura 2.3 Representación gráfica de una entidad

2.2.2 Asociación

Una vez que se cuenta con las entidades, es importante establecer enlaces entre ellas. El concepto de *asociación* sirve a este propósito. La asociación se puede ver como una clase genérica de enlaces entre objetos pertenecientes a las entidades involucradas.

Ejemplo. Las siguientes afirmaciones

*Pedro y María poseen un automóvil
Juan posee un camión*

son enlaces entre objetos, de los cuales unos pertenecen a una entidad PERSONA y otros a una entidad VEHICULO. Así, se puede establecer la asociación (posee) entre las entidades PERSONA y VEHICULO, según se aprecia en la figura 2.4.

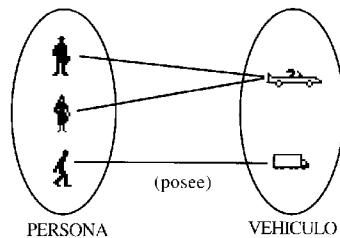


Figura 2.4 Ejemplo de una asociación

Si una asociación es entre dos entidades se dice que es *binaria*. Si la asociación involucra a n entidades, se dice que es *n-aria*.

En la práctica del diseño de una base de datos, es difícil encontrar situaciones que ameriten la representación por medio de una asociación entre más de tres entidades.

Ejemplo. En la figura 2.5, se muestra un caso de una asociación de grado 3 o terciaria. Aquí, la asociación (tratamiento) representa el o los medicamentos que son recetados por uno o varios médicos a un determinado paciente. En el capítulo 6, dedicado al diseño de bases de datos, se mostrará que ante una situación como ésta es recomendable crear una nueva entidad, por ejemplo TRATAMIENTO, y establecer tres asociaciones entre las entidades originales y la nueva entidad:

MEDICO (prescribe el) TRATAMIENTO
PACIENTE(sigue el) TRATAMIENTO
MEDICAMENTO (sirve para el) TRATAMIENTO

Según los objetos enlazados entre asociaciones, como se aprecia en la figura 2.6, se puede hablar de cuatro tipos de asociaciones:

- Asociaciones *uno a uno* (1-1)
- Asociaciones *uno a muchos* (1-N)

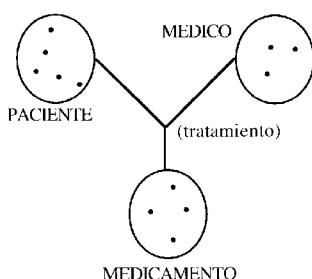


Figura 2.5 Ejemplo de una asociación terciaria

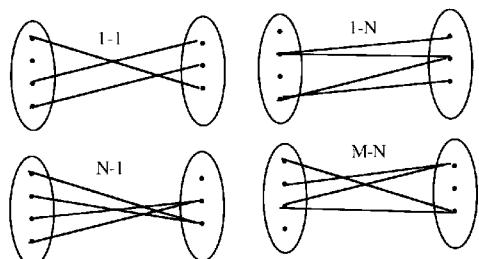


Figura 2.6 Posibles asociaciones entre entidades

- Asociaciones *muchos a uno* (N-1)
- Asociaciones *muchos a muchos* (M-N).

Ejemplo. Considerar las siguientes tres entidades: PROFESOR, ESTUDIANTE y DEPARTAMENTO.

Entre las asociaciones que se pueden establecer entre estas entidades están las siguientes:

PROFESOR (es director de) DEPARTAMENTO, es una asociación 1-1, pues cada departamento cuenta con un solo director y cada director lo es de un solo departamento.

DEPARTAMENTO (cuenta con) PROFESOR, es una asociación 1-N, pues un departamento puede tener varios profesores y, en principio se supone que cada profesor trabaja en un solo departamento.

ESTUDIANTE (se matricula en) DEPARTAMENTO, es una asociación N-1, pues varios estudiantes se matriculan en un departamento y cada uno de ellos lo hace en uno solo. PROFESOR (enseña a) ESTUDIANTE es una asociación M-N, ya que un profesor puede enseñar a varios estudiantes y un estudiante puede recibir la enseñanza de varios profesores.

En la figura 2.7 se muestran las representaciones gráficas de los cuatro tipos de asociaciones binarias que se definieron previamente.

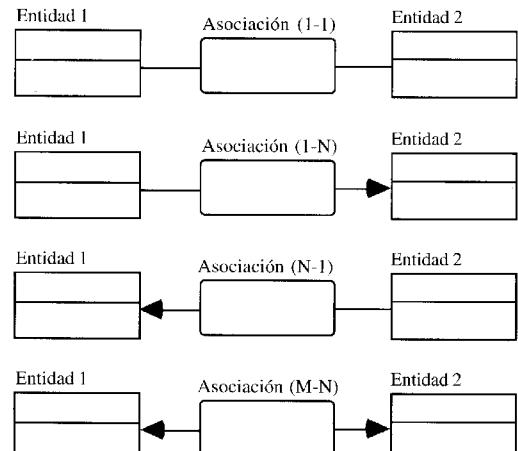


Figura 2.7 Representación gráfica de los diferentes tipos de asociaciones

Es importante mencionar que el tipo de asociación traduce una restricción del mundo real, que puede considerarse como una regla de integridad que debe respetarse al interior del sistema.

En efecto, si se considera la asociación

PROFESOR (es director de) DEPARTAMENTO

del ejemplo anterior, se observa que es una asociación 1-1. Esto garantiza que, bajo estas circunstancias, no es posible que un mismo profesor sea director de dos departamentos y que cada departamento debe contar necesariamente con un director.

Para afinar aun más el concepto de asociación entre objetos de dos entidades, se puede introducir la noción de cardinalidad.

La *cardinalidad* de una asociación con respecto a una entidad se puede definir como un par (x,y) tal que:

- x representa el número mínimo de objetos asociados con un objeto dado de la entidad
- y representa el número máximo de objetos asociados con un objeto dado de la entidad.

Ejemplo. Considerar las entidades PROFESOR y ESTUDIANTE y la asociación (enseña), según se presenta en la figura 2.8.

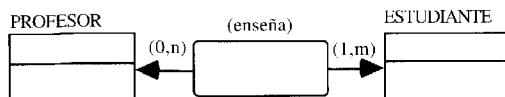


Figura 2.8 Ejemplo de cardinalidades de una asociación

En este caso, un profesor puede no impartir lecciones. Por lo tanto su cardinalidad mínima es 0. Asimismo, un profesor puede impartir lecciones a varios estudiantes, en cuyo caso el máximo es n.

Por otra parte, una persona para ser considerada estudiante debe estar matriculada al menos con un profesor y puede recibir la enseñanza de varios profesores. Así, el mínimo sería 1 y el máximo m.

Es importante destacar que tanto la notación sobre el tipo de una asociación, como la determinación de su cardinalidad parecieran conceptos redundantes. En efecto, éstos lo son hasta cierta medida. Sin embargo, se ha hecho la diferencia pues se considera que ayudan a visualizar mejor el significado de la asociación.

2.2.3 Atributo

El atributo es uno de los principales medios usados en los modelos semánticos, y otros modelos, para reunir y caracterizar objetos.

Un *atributo* sirve para representar ciertas características genéricas de una entidad. Es una propiedad distintiva de una entidad o asociación.

Por ejemplo, si se considera una aplicación sobre las inscripciones de vehículos, dos entidades pueden ser PROPIETARIO y VEHICULO. Para el caso de la entidad PROPIETARIO, ésta puede caracterizarse por medio de los siguientes atributos: Nombre del propietario, Dirección del propietario, Tipo de licencia, etc.

Los atributos que se le asignen a una entidad dependen del contexto de la aplicación. Un atributo de un propietario puede ser su afiliación política o su edad. Sin embargo, en la aplicación anterior, estos atributos no son pertinentes.

En la figura 2.9 se muestra la ubicación de los atributos al interior de las entidades.

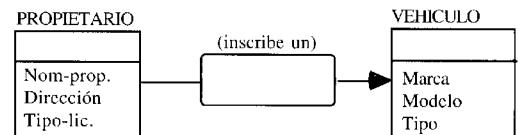


Figura 2.9 Representación gráfica de los atributos

Es importante notar que un atributo es un conjunto de posibles valores. No se debe confundir el nombre de un atributo con su valor. Además, varios objetos de una entidad o de una asociación pueden tener un mismo valor para un atributo dado. La identificación de los objetos es muy importante. Además, para que una entidad exista, al menos debe tener un atributo. Un atributo que representa una característica precisa solo se encuentra una vez en una entidad. Siempre se deben dar nombres no ambiguos a los atributos.

También hay que destacar que una asociación puede tener atributos ligados a ella. En el caso anterior, la asociación puede tener como atributos, la fecha de la inscripción, así como el empleado que realizó dicha inscripción.

Por otra parte, se debe mencionar que un concepto particular puede, según el contexto, representar un atributo, una asociación o una entidad, según se muestra en el siguiente ejemplo.

Ejemplo. Considerar las dos entidades que se presentan en la figura 2.10.

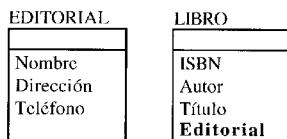


Figura 2.10 Ejemplo de dos entidades

El concepto *Editorial* ha sido utilizado en un caso como entidad y en el otro como atributo de una entidad.

Una vez introducido el concepto de atributo, se debe mencionar que algunos de ellos pueden identificar en forma única los diferentes objetos de la entidad. Este conjunto mínimo se llama *llave*.

Ejemplo. Si se consideran las entidades de la figura 2.9, un atributo como el Número de cédula puede ser una llave de la entidad PROPIETARIO, pues se sabe que cada persona tiene un único número de cédula que lo identifica. El nombre de un propietario no puede ser llave pues existen personas diferentes con el mismo nombre. Por su parte, con respecto a la entidad VEHICULO, una llave puede ser la placa del vehículo.

Una entidad puede tener varias llaves asociadas y las llaves pueden tener más de un atributo. Cada una de las llaves de una entidad se denominan *llaves candidatas*. De estas posibles llaves, la que se escoge como principal se denomina *llave primaria* y las otras se convierten en *llaves alternas*.

En la figura 2.11 se puede apreciar cómo la llave primaria de las entidades, se coloca en el recuadro superior de la caja que representa tal entidad.

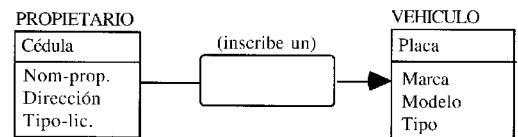


Figura 2.11 Ubicación de las llaves primarias

Ejemplo. Sea la entidad CURSO, la cual brinda información de las materias que se imparten en una universidad y que está formada de los siguientes atributos: Código-curso, Código-materia, Número-grupo, Número-profesor, Aula, Hora, Día. En este caso se pueden establecer varias llaves candidatas:

- Código-curso
- Código-materia, Número-grupo
- Aula, Hora, Día
- Número-profesor, Día, Hora.

Si se escoge a Código-curso como llave primaria, las otras pasan a ser llaves alternas. Si un atributo forma parte de la llave alterna n, se puede poner junto al atributo dicho valor, entre corchetes, según se aprecia en la figura 2.12.

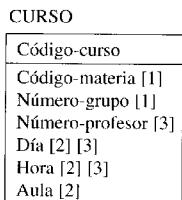


Figura 2.12 Ejemplo de llaves primarias y alternas

2.2.4 Herencia

Una forma de administrar la complejidad de un sistema es por medio de la *herencia* o *generalización-especialización* y consiste en afinar una entidad en otras más específicas. Este afinamiento no es más que una partición de una entidad que se hace para puntuizar aun más las características de los objetos. Así, si se tiene una entidad ARTISTA, que represente a las personas destacadas en las artes de un determinado país, es posible que se desee contar con una mayor información sobre estas personas en sus diferentes áreas.

Así por ejemplo, se podría partitionar la entidad ARTISTA, que se denomina *super-entidad*, en las entidades MUSICO, PINTOR, ESCULTOR, ACTOR, y que se llamarán *sub-entidades*, según se aprecia en la figura 2.13.

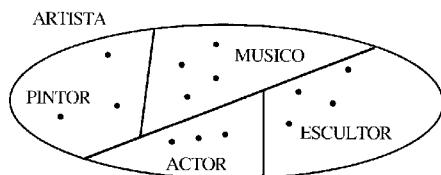


Figura 2.13 Ejemplo de partición de una entidad

La generalización-especialización se materializa vía el predicado “*es un*”. Así, un pintor *es un* artista, un actor *es un* artista, etc.

En la figura 2.14 se da una representación gráfica de una generalización-especialización.

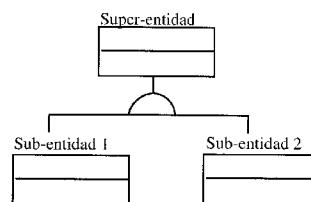


Figura 2.14 Representación gráfica de una herencia

Ejemplo. Sea la entidad PERSONA de los funcionarios y estudiantes de una Universidad. Se pueden definir sub-entidades como ACADEMICO, ADMINISTRATIVO, ESTUDIANTE de la super-entidad PERSONA. Asimismo, se puede ver la entidad ACADEMICO como super-entidad de las entidades DOCENTE e INVESTIGADOR, según se aprecia en la figura 2.15.

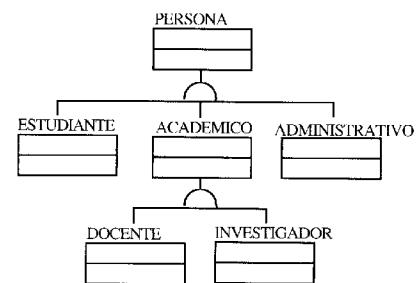


Figura 2.15 Ejemplo de herencia

Sea E una super-entidad y F una sub-entidad de E. Entonces, F hereda todas las propiedades de E, es decir, si F es un E, entonces hereda todos los atributos de E. Por otra parte, los atributos adicionales -propios- de F, no pueden ser heredados por E. En una herencia, como los objetos de la super-entidad y de sus sub-entidades son los mismos del mundo real, la llave primaria de la super-entidad se hereda a las sub-entidades y será la llave primaria de las mismas.

Ejemplo. En la figura 2.15, los atributos de las entidades PERSONA y ESTUDIANTE pueden ser las que se presentan en la figura 2.16.

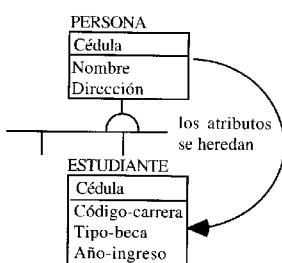


Figura 2.16 Ejemplo de atributos involucrados en una herencia

En este caso, el conjunto de atributos que caracteriza a la entidad ESTUDIANTE es la unión de los atributos de PERSONA y sus propios atributos.

Sin embargo, se debe mencionar que una herencia no siempre es una partición. Así, es posible tener objetos que pertenezcan a varias sub-entidades. Cuando se tiene una herencia de este tipo se denomina *categoría* y su representación gráfica se muestra en la figura 2.17.

Ejemplo. Siguiendo con el ejemplo anterior, si se tiene la super-entidad PERSONA y las sub-entidades ESTUDIANTE y FUNCIONARIO y se permite que un funcionario pueda a su vez ser estudiante de la Universidad, entonces la herencia no establece una partición.

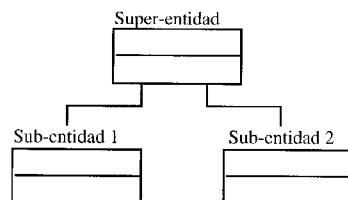


Figura 2.17 Representación de una categoría

Asimismo, es posible definir el concepto de *herencia múltiple*, esto es, que una sub-entidad puede tener varias super-entidades asociadas.

Sin embargo, este concepto no será tratado en este capítulo pues queda fuera de los alcances del modelo; más bien, será estudiado en el capítulo 14, en donde se define el paradigma de la orientación a objetos.

2.2.5 Composición

Otra herramienta que se utiliza para administrar la abstracción y la complejidad en los sistemas se conoce como *composición* -o agregación-.

En una composición, la idea es que se pueden reunir varias entidades de un mismo nivel en una nueva entidad de un nivel superior. En este caso, la entidad de nivel superior llamada el *compuesto*, se determina por medio de las entidades del nivel inferior, llamadas los *componentes*.

Ejemplo. En la figura 2.18 se muestra cómo la salsa china se compone de una serie de componentes o ingredientes; todos juntos en las cantidades adecuadas, constituyen la salsa china.

Así, una entidad, compuesta de otras entidades, se podría definir como el producto cartesiano de estas entidades.

En la figura 2.19 se da la representación gráfica de una composición.

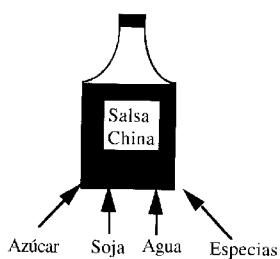


Figura 2.18 Ejemplo de una composición

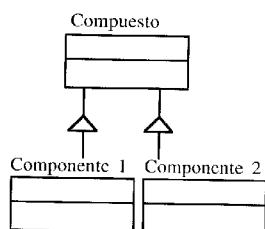


Figura 2.19 Representación gráfica de una composición

Análogamente, como en el caso de las asociaciones, se pueden definir cardinalidades entre el compuesto y sus componentes.

Ejemplo. Si se considera una entidad AVION, que representa los aviones en construcción de una compañía de aviones, ésta puede ser constituida a partir de otras entidades, por ejemplo: MOTOR, ASIENTO y FUSELAJE, según se aprecia en la figura 2.20.

En este caso se tiene que un avión está compuesto de tres módulos principales, a saber: motores, asientos y fuselaje. Según la figura un avión puede tener al menos

un motor y a lo sumo cuatro motores; al menos 150 asientos y a lo sumo 300 asientos y solo un fuselaje. Por su parte, la cardinalidad del componente motor dice que un motor puede no estar instalado en un avión y que a lo sumo estará en un avión. Lo mismo ocurre con las entidades componentes ASIENTO y FUSELAJE.

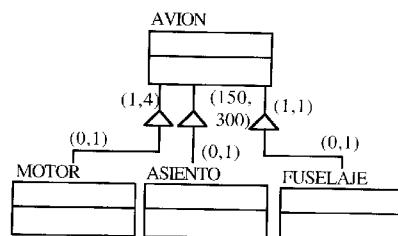


Figura 2.20 Ejemplo de una composición

Una composición puede traducirse por medio de la expresión “*es parte de*”. Así por ejemplo, un asiento *es parte de* un avión.

Con respecto a las llaves para cada una de las entidades en una composición, en el capítulo 6 referente a la metodología de diseño, se explica cómo traducir una composición en relaciones mediante la creación de una nueva relación, en donde la llave primaria es la del compuesto y los atributos no llave serán las llaves primarias de los componentes.

Para finalizar, en la figura 2.21, se muestra un esquema conceptual que representa el alquiler de vehículos de una Compañía dedicada a esta actividad. En el capítulo 6, se explican los pasos que se deben seguir para llegar a tal esquema.

EJERCICIOS Y PREGUNTAS DE REPASO

- 2.1 Definir en forma general lo que significa un modelo de datos.
- 2.2 ¿Qué caracteriza a un modelo de datos semántico?

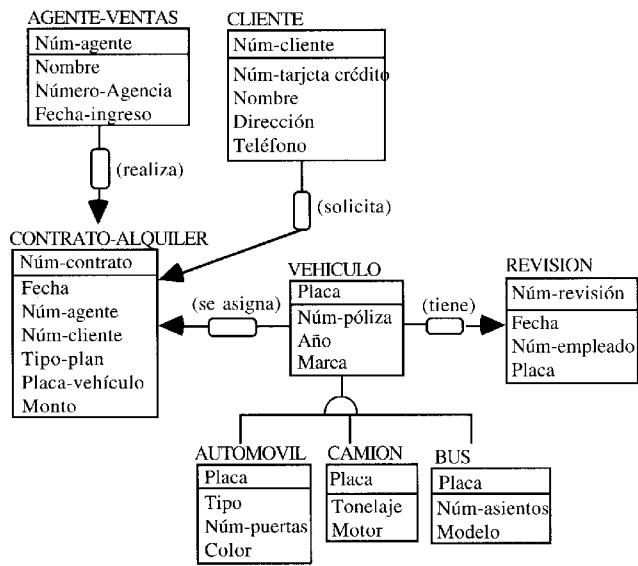
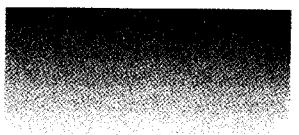


Figura 2.21 Ejemplo de un esquema conceptual

- 2.3. Nombrar un modelo de datos semántico.
- 2.4. ¿Cuáles son los elementos principales que conforman el modelo de datos presentado?
- 2.5. ¿Cuáles son los diferentes tipos de asociación?
- 2.6. ¿Qué es la cardinalidad de una asociación?
- 2.7. ¿Por qué en una herencia, la super-entidad y las sub-entidades deben tener la misma llave primaria?

- 2.8. Según el modelo de datos semántico estudiado, ¿cuáles son las dos formas de manejar la abstracción y la complejidad de un sistema?
- 2.9. Definir el concepto de composición y dar varios ejemplos en que este concepto se puede aplicar.



3

Modelo Relacional de Codd

«Quienes no piensan por cuenta propia concluyen por ser hombres de color de niebla; ignoran que el hombre esculpe su imagen en todas las obras de la creación que se han detenido por algunos instantes en sus reflexiones. Una inteligencia incrustada de tradiciones, de convenciones, de opiniones hechas, de intolerancias y dogmatismos, como el río sembrado de grandes piedras, no permite la navegación de la visión trascendente. Preciso es que un constante pensar, ya metódico o ya tempestuoso, limpie de sirtes el entendimiento.»

Roberto Brenes Mesén (1874-1947),
educador y ensayista costarricense.

3.1 INTRODUCCION

Como se mencionó anteriormente, las ideas fundamentales del modelo de datos relacional aparecieron, en forma pública, en un artículo de E. Codd de 1970 [CODD70]. Este artículo, para muchos, representa el hito más importante en la historia de la tecnología de bases de datos.

Desde la óptica del modelo relacional de Codd, la base de datos se puede ver como un conjunto de estructuras llamadas *relaciones* y en donde los datos se organizan en forma tabular.

Una de las grandes virtudes del modelo relacional es su facilidad de uso, que ha permitido que personas que no son profesionales en informática, hoy puedan construir sus propias bases de datos y hacerlas con lenguajes no procedimentales.

Tal es la contribución de Codd a la tecnología de bases de datos. Hoy en día, la industria de las bases de datos relacionales se calcula en miles de millones de dólares por año.

Una de las características de los SABDs basados en este modelo es la simplicidad de uso, sin por ello, dejar de ser poderosos y confiables.

Este acercamiento de la tecnología de bases de datos relacionales a otros profesionales se puede resumir en los siguientes puntos [MIRA88]:

- simplicidad de los conceptos de base,
- operadores de manipulación muy poderosos,
- sustentada sobre una sólida teoría matemática.

Como en todo modelo de datos, se estudiará su estructura, las operaciones de manipulación, así como las reglas de integridad que validan la permanencia o no de un conjunto de datos dentro del sistema.

3.2 ESTRUCTURA DEL MODELO

El eje principal del modelo relacional reposa sobre el concepto de relación. Antes de dar una definición formal de este concepto, se hará una presentación intuitiva.

En una base de datos relacional, los usuarios tienen una visión de ésta de tipo tabular, es decir, los datos se organizan en forma matricial.

Ejemplo. Si se tiene una base de datos que implementa un sistema de información sobre las ventas de los productos de una empresa, una de las entidades principales de este sistema será la entidad CLIENTE y ésta puede verse como la relación que se muestra en la figura 3.1.

CLIENTE			
Número	Nombre	Dirección	Teléfono
1243	Juan Mora	Alajuela	4424405
2000	Ana Coto	Heredia	2378900
1894	Ana Coto	Cartago	5528434

Figura 3.1 Ejemplo de una relación

En este caso se tiene una relación CLIENTE con cuatro atributos: Número, Nombre, Dirección y Teléfono. Además, consta de tres tuplas:

[1243, Juan Mora, Alajuela, 4424405], [2000, Ana Coto, Heredia, 2378900] y [1894, Ana Coto, Cartago, 5528434].

Por otra parte, se definen los dominios, es decir, los valores que puede tomar cada uno de los atributos. Por ejemplo, el dominio del atributo Teléfono se puede definir como el conjunto de números naturales de siete dígitos y denotado por $\text{dom}(\text{Teléfono}) = \{ n / n \text{ es un número natural con 7 dígitos} \}$.

3.2.1 Conceptos básicos

Tradicionalmente, en el modelo relacional, una relación R se define como un subconjunto de un producto cartesiano de n dominios D_1, \dots, D_n , es decir,

$$R \subseteq D_1 \times D_2 \times \dots \times D_n$$

Sin embargo, en este libro se hará una introducción diferente de este concepto, por razones que se discutirán luego.

En efecto, en primer lugar, se define el concepto de armazón. Un *armazón* ($m \times n$) se define como un arreglo de $m+1$ filas por n columnas, según se aprecia en la figura 3.2.

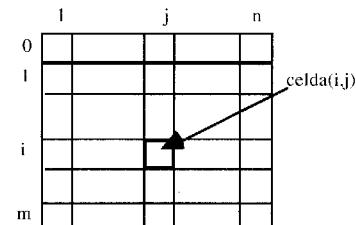


Figura 3.2 Diagrama de un armazón

Por su parte, un *dominio* es un tipo de dato, como puede ser INTEGER, FLOAT, etc.

Además, un *atributo* se define como la identificación que se le da a un conjunto de valores pertenecientes a un tipo de dato -dominio- dado.

A continuación, se presentan las reglas que se deben satisfacer cuando se llenan las celdas de un armazón, con el fin de definir el concepto de relación.

- Cada celda $c(0, j)$ estará constituida de un par (A_j, D_j) , en donde A_j es un atributo y D_j es un dominio sobre el cual se define el atributo.
- Los atributos y los dominios se designarán por palabras que inician con mayúscula.
- Los atributos deben ser diferentes, no así los dominios. Como el concepto de dominio es una noción semántica, entonces varios atributos pueden compartir el mismo dominio.

- Cada celda $c(i,j)$, se llama *componente* y toma sus valores del dominio D_j .
- Cada fila $[c(i, 1), c(i, 2), \dots, c(i, n)]$, $1 \leq i \leq m$, se denomina *tupla*.
- Las tuplas en un armazón deben ser diferentes, esto significa que deben diferenciarse al menos en los valores de un mismo atributo.

Así, con estas reglas se define una *relación R de grado n*, como un armazón $m \times n$ completamente utilizado, con $m, n \geq 1$, según se aprecia en la figura 3.3.

	i	j	n
0	(A_1, D_1)	(A_j, D_j)	(A_n, D_n)
1	a_{11}	a_{1j}	a_{1n}
i	a_{ij}	a_{ij}	a_{in}
m	a_{m1}	a_{mj}	a_{mn}

Figura 3.3 Definición de relación

Según esta definición, una relación se puede ver como constituida de dos partes. En primer lugar, la fila 0, que se llamará *encabezado* de la relación, y que se denota por

$$R((A_1, D_1), (A_2, D_2), \dots, (A_n, D_n))$$

Y en segundo lugar, el conjunto de las tuplas de la relación, que se llamará *contenido* de la relación, y se denota por

$$\{ [a_{11}, \dots, a_{1n}], [a_{21}, \dots, a_{2n}], \dots, [a_{m1}, \dots, a_{mn}] \}$$

Con el fin de ser consecuentes con las notaciones que utilizan la mayoría de los autores, se debe notar que el encabezado se conoce también como *esquema de la relación* y se denotará sin los dominios respectivos, de la siguiente forma:

$$\underline{R}(A_1, A_2, \dots, A_n)$$

Por su parte, cuando se haga referencia a una relación, simplemente se denotará por

$$R(A_1, A_2, \dots, A_n)$$

Puesto que una relación es en realidad un *conjunto* de elementos (tuplas), ésta puede definirse según dos enfoques: uno, desde un punto de vista *extensivo* y otro, desde un punto de vista *intensivo*.

Así, una relación R en *extensión* se define dando el conjunto de todas sus tuplas, es decir, dando el contenido de la misma.

La relación R también se puede definir en *intensión* dando el predicado de pertenencia $P(x_1, x_2, \dots, x_n)$ de una tupla en R tal que

$$R = \{ [a_1, a_2, \dots, a_n] / P(a_1, a_2, \dots, a_n) \text{ es verdadero} \}$$

Ejemplo. Sea la relación de la figura 3.1. El encabezado

CLIENTE(Número, Nombre, Dirección, Teléfono)

se puede definir como el predicado

$P(n, c, d, t) : "el cliente de nombre c, tiene como número de cliente n, vive en la dirección d y tiene como número de teléfono t".$

Así, como el cliente de nombre Juan Mora, tiene como número de cliente 1243, vive en la dirección Alajuela y tiene como número de teléfono 4424405, el predicado $P(1243, Juan\ Mora, Alajuela, 4424405)$ es verdadero y por lo tanto

$[1243, Juan\ Mora, Alajuela, 4424405]$ es una tupla de la relación CLIENTE.

Es importante mencionar que una relación, desde el punto de vista matemático no es conmutativa, es decir, al ser un subconjunto de un producto cartesiano, no se pueden permutar dos conjuntos de dicho producto pues se tendrían resultados diferentes.

En cambio, en el modelo relacional, se pueden permutar atributos obteniendo relaciones equivalentes.

Ejemplo. Si se considera la relación

CLIENTE(Número, Nombre, Dirección, Teléfono),

también puede presentarse bajo la siguiente forma:

CLIENTE(Teléfono, Número, Dirección, Nombre).

En efecto, la tupla [4424405, 1243, Alajuela, Juan Mora] se encuentra en dicha relación y el predicado significaría: *la persona con el teléfono 4424405 y cuyo número de cliente es 1243 y vive en Alajuela, se llama Juan Mora*, que es equivalente al del ejemplo anterior.

Con el fin de contar con un ejemplo sobre el cual se puedan explicar los diferentes conceptos de este capítulo y del libro en general, se introducen tres relaciones que traducen los viajes que realizan anualmente los miembros de un Club de Ecoturismo a diferentes sitios ecológicos del planeta. En la figura 3.4 se muestran los predicados que generan estas relaciones y algunas posibles tuplas.

- TURISTA = { [nt, t, p] / el turista número nt y de nombre t es del país p }.
- SITIO = { [ns, s, t, c] / el sitio número ns y de nombre s es del tipo t y se encuentra en el continente c }.
- VIAJE = { [nv, nt, ns, fs, fl, cs] / el viaje número nv lo hace el turista número nt al sitio número ns con fecha de salida fs y fecha de llegada fl, y sale de la ciudad cs }.

TURISTA

Número -Turista	Nombre -Turista	País
300	Carlos	Costa Rica
301	Pierre	Francia
302	John	Jamaica
303	Mario	Panamá
304	Alí	Túnez

Fig. 3.4 continúa...

SITIO

Número -Sitio	Nombre -Sitio	Tipo	Continente
125	Isla Moorea	Mar/Litoral	Oceanía
126	Bahía Matsushima	Mar/Litoral	Asia
127	Irazú	Volcán	América
128	Ngorongoro	Volcán	Africa
129	Valle de la Muerte	Desierto	América
130	Kilimandjaro	Volcán	Africa

VIAJE

Número -Viaje	Número -Turista	Número -Sitio	Fecha -Salida	Fecha -Llegada	Ciudad -Salida
03-96	301	125	03/03/96	03/10/96	París
04-96	303	129	07/04/96	07/14/96	Las Vegas
05-96	301	128	07/05/96	07/12/96	Dar-es-Salam
06-96	304	127	07/06/96	07/14/96	San José
07-96	302	128	07/28/96	08/13/96	Mombasa

Figura 3.4 Ejemplo de la base de datos del Club de Ecoturismo

3.2.2 Llave primaria, llave externa

Sea el encabezado $R(A_1, A_2, \dots, A_n)$ de una relación R . Algunos atributos pueden determinar el conocimiento de los atributos restantes.

Un conjunto mínimo de atributos que determinan al resto de atributos se denomina *llave candidata*. De estas llaves candidatas, la que se escoga como llave principal -es una escogencia arbitraria- se denomina *llave primaria* y al resto de llaves candidatas se les llamará *llaves alternativas*.

Por otra parte, se dice que un dominio es *primario* si sobre él se ha definido una llave primaria.

Una *llave externa* es un atributo definido sobre un dominio primario y que no es llave primaria. Estas definiciones son compatibles con las hechas sobre las llaves en el modelo semántico.

En el capítulo 6 se estudiará el traslado de las entidades del modelo semántico a relaciones, sin embargo, se puede adelantar el hecho de que una llave externa traducirá la asociación entre las dos entidades -relaciones- involucradas.

Ejemplo. Sean las relaciones

PROFESOR(Número-Profesor, Cédula, Nombre-Profesor, Título, Categoría) y

ESTUDIANTE(Carné, Nombre, Nombre-Carrera, Número-Profesor),

que traducen la lista de profesores de una Universidad que imparten cursos a los estudiantes que se han matriculado en el año lectivo. En la figura 3.5, se da un ejemplo de estas relaciones.

PROFESOR

Número -Profesor	Cédula	Nombre -Profesor	Título	Categoría
2234	2302445	Juan Mora	M.Sc.	Asociado
2356	1456989	Mario Coto	Ph.D.	Asociado
2678	5435433	Ana Salas	Dr.	Catedrático

ESTUDIANTE

Carné	Nombre -Carrera	Nombre -Profesor	Número
878987	Astrid Mata	Electrónica	2234
852341	Julia Brenes	Computación	2678
912348	Pedro Sancho	Computación	2678

Figura 3.5 Ejemplo de relaciones

En este caso, se puede definir el atributo Número-Profesor en la relación PROFESOR como llave primaria y Cédula como llave alterna, pues cada uno de estos atributos determina de forma única al resto de los atributos.

Por su parte, en la relación ESTUDIANTE, se tiene el atributo Carné como llave primaria y el atributo Número-Profesor como llave externa. Este último atributo traduce la asociación 1-N que existe entre las relaciones PROFESOR y ESTUDIANTE.

Para diferenciar los atributos que conforman la llave primaria de los que no están en ella, éstos se deben subrayar.

3.3 REGLAS DE INTEGRIDAD DEL MODELO

3.3.1 Integridad de dominio

La *integridad de dominio* se introduce para controlar la sintaxis y la semántica de un dato cualquiera y concierne al tipo de definición del dominio. Así por ejemplo, según la figura 3.4, los valores de Número-Sitio pueden establecerse sobre el tipo de dato o dominio {100, 101, ..., 999 }.

Una idea interesante sobre los dominios es el hecho de que cuando se definen los atributos de una relación, algunos de ellos se establecen sobre dominios equivalentes. En la gran mayoría de los SABDs actuales al definir las características de los atributos, éstas se hacen al interior de la definición de la relación. Sin embargo, debe pensarse en una alternativa explícita, esto es, establecer dominios como tipos de datos y definir los atributos sobre éstos, que se encontrarían al exterior de la definición de las relaciones. Esto permitiría rescatar una mayor semántica de los atributos y permitiría comparaciones entre atributos al interior de una relación o entre atributos de varias relaciones, más eficientes y consistentes.

Ejemplo. Si se considera la relación

VIAJE(Núm-Viaje, Núm-Turista, Núm-Sitio, Fec-Salida, Fec-Llegada, C-Salida) podría describirse de la siguiente forma:

Definición de dominios:

Número: Número de 5 dígitos.

Fecha: Fecha definida con un formato 'date' -mm/dd/aa-

Ciudad Ciudad en un país y establecida como un "string" de 20 caracteres.

Definición de la relación :

VIAJE (Núm-Viaje, *definido sobre* Número,
 Núm-Turista, *definido sobre* Número,
 Núm-Sitio, *definido sobre* Número,
 Fec-Salida *definido sobre* Fecha,
 Fec-Llegada, *definido sobre* Fecha,
 C-Salida, *definido sobre* Ciudad)

Así, al establecer un conjunto centralizado de dominios, éstos pueden ser utilizados por atributos diferentes al interior de una misma relación, o incluso entre atributos de relaciones diferentes.

3.3.2 Integridad de relación

La *integridad de relación* por su parte, se refiere a los valores de los atributos que conforman la llave primaria.

Cada relación posee exactamente una llave primaria. Cada valor de una llave primaria identifica la tupla en forma única. De ahí que la integridad de relación garantiza que los valores de los atributos que conforman la llave primaria deben ser *únicos* y no se deben permitir los valores nulos -un valor nulo es un carácter especial que representa el conjunto vacío-.

Por razones de diseño y de rendimiento, hasta donde sea posible, se recomienda que la llave primaria de cada relación esté constituida por un solo atributo.

3.3.3 Integridad referencial

La *integridad referencial* se refiere a la restricción que debe darse entre los valores de los atributos que conforman la llave primaria de una relación primaria y los valores que estos atributos pueden tomar como llave externa en una relación secundaria.

Ejemplo. Si se considera la base de datos de la figura 3.1, se diría que las relaciones SITIO y TURISTA son *primarias*, pues sus existencias no dependen de ninguna

otra relación. En cambio, la existencia de la relación VIAJE depende de las relaciones anteriores. Por esta situación, se dice que es *secundaria*.

Así, el conjunto de valores que toma el atributo Número-Sitio en la relación VIAJE, debe ser un subconjunto de los valores del atributo Número-Sitio en la relación SITIO. Asimismo, el conjunto de valores de Número-Turista en la relación VIAJE debe ser un subconjunto de los valores que aparecen en el atributo Número-Turista de la relación TURISTA.

A la hora de escoger un SABD relacional un factor importante en la decisión final es la capacidad que tenga el producto para brindar facilidades de establecer este tipo de reglas de integridad.

Como tercer gran tema que se debe estudiar en un modelo de datos en general, y en el relacional en particular, es el referente a la manipulación de los datos contenidos en la base de datos.

Con respecto al modelo relacional se vieron dos formas de definición. Por lo tanto, también se puede hablar de dos grandes tipos de lenguajes de manipulación de datos:

- El lenguaje algebraico, basado en expresiones algebraicas cuyos elementos son relaciones y operadores
- Los lenguajes predicativos, que se basan en la lógica de primer orden o de predicados, y que a su vez se dividen en predicativos de tuplas y predicativos de dominios,

Los diferentes tipos de lenguajes relationales se estudiarán en la próxima sección.

3.4 ALGEBRA RELACIONAL

El álgebra relacional está constituida de una serie de operadores que se aplican a las relaciones de la base de datos.

Una consulta de usuario que se haga por medio de un lenguaje de este tipo, se interpreta como una expresión algebraica constituida por relaciones de base, vistas y operadores.

Una tal expresión se puede visualizar como un árbol, en el sentido de la teoría de grafos. En efecto, las hojas de dicho árbol serán las relaciones involucradas en la consulta, los nodos del árbol serán los operadores y, finalmente, la raíz representará la relación resultado.

Es por esta razón que, al estudiar cada uno de los operadores, se van a introducir sus representaciones gráficas.

Son ocho los operadores del álgebra relacional y se clasifican en operadores relacionales y operadores conjuntistas.

Por su parte, los operadores relacionales se pueden clasificar, a su vez, como operadores *unarios* -aplicados sobre una sola relación- y *binarios* -aplicados sobre dos relaciones-, según se puede apreciar en la figura 3.6.

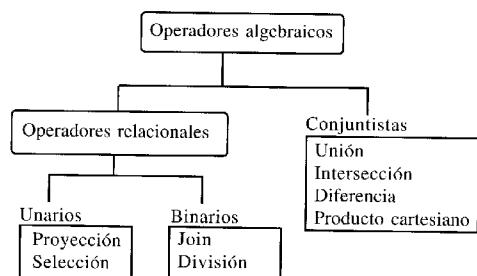


Figura 3.6 Los operadores algebraicos

3.4.1 Operadores conjuntistas

Sean $R(A_1, A_2, \dots, A_n)$ y $S(B_1, B_2, \dots, B_n)$ dos relaciones de grado n y donde los atributos A_i y B_i , $i \in \{1, \dots, n\}$, se definen sobre los mismos dominios, es decir son comparables. Entonces, los operadores de unión, intersección y diferencia, se definen según la teoría de conjuntos.

- *unión* $R \cup S = \{ t / t \in R \vee t \in S \}$
- *intersección* $R \cap S = \{ t / t \in R \wedge t \in S \}$
- *diferencia* $R - S = \{ t / t \in R \wedge t \notin S \}$

Otro operador conjuntista es el producto cartesiano en donde los atributos de las relaciones involucradas pueden definirse sobre cualquier dominio, y su definición es la siguiente:

- *producto cartesiano* $R \times S = \{ [t,s] / t \in R \wedge s \in S \}$

En la figura 3.7 se simboliza el efecto de aplicar estos operadores conjuntistas a dos relaciones R y S .

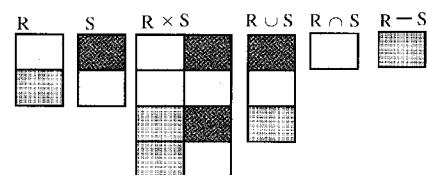


Figura 3.7 Resultado de aplicar los operadores conjuntistas

En el caso de los operadores conjuntistas, sus representaciones gráficas se muestran en la figura 3.8.

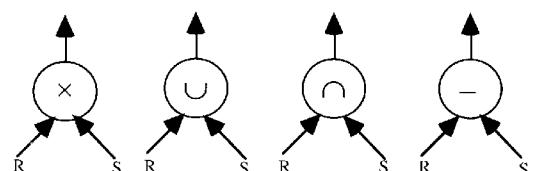


Figura 3.8 Representación gráfica de los operadores conjuntistas

Ejemplo. Sean las dos relaciones

DEPORTIVA(Número, Nombre, Dirección, Teléfono) y
CULTURAL(Código, Nombre-Estudiante, Dirección, Teléfono-Estudiante)

que representan los estudiantes matriculados en actividades culturales o deportivas de una Universidad.

Si se desea tener la lista de estudiantes que se encuentran matriculados en al menos una actividad, se puede hacer una unión entre las dos relaciones DEPORTIVA \cup CULTURAL.

Por su parte, la lista de los estudiantes que se encuentran matriculados en ambas actividades se podría conseguir haciendo una intersección entre ambas relaciones DEPORTIVA \cap CULTURAL.

También, la diferencia DEPORTIVA — CULTURAL daría la lista de estudiantes que llevan solo actividades deportivas, independientemente de si se encuentran matriculados o no en actividades culturales. Finalmente, el producto cartesiano dará todas las posibles combinaciones de tuplas de las dos relaciones. La figura 3.9 muestra un posible ejemplo de estas situaciones.

En el caso de una unión, intersección o diferencia, se va a suponer que el atributo que aparece en el resultado es el de la relación que se encuentra en la parte izquierda.

DEPORTIVA

No	Nombre	Dirección	Teléfono
124	Juan Coto	Alajuela	4427867
149	Ana Salas	Heredia	2375643
267	Eva Mora	Heredia	2379898

CULTURAL

No	Nombre	Dirección	Tel-estud
124	Juan Coto	Alajuela	4427867
569	Eric Soto	Limón	7679899

DEPORTIVA \cup CULTURAL

No	Nombre	Dirección	Teléfono
124	Juan Coto	Alajuela	4427867
149	Ana Salas	Heredia	2375643
267	Eva Mora	Heredia	2379898
569	Eric Soto	Limón	7679899

DEPORTIVA \cap CULTURAL

No	Nombre	Dirección	Teléfono
124	Juan Coto	Alajuela	4427867

Fig. 3.9 continúa...

DEPORTIVA — CULTURAL

No	Nombre	Dirección	Teléfono
149	Ana Salas	Heredia	2375643
267	Eva Mora	Heredia	2379898

DEPORTIVA \times CULTURAL

No	Nombre	Dirección	Teléfono	Co	Nombre	Dirección	Teléfono
124	Juan Coto	Alajuela	4427867	124	Juan Coto	Alajuela	4427867
149	Ana Salas	Heredia	2375643	569	Eric Soto	Limón	7679899
267	Eva Mora	Heredia	2379898	124	Juan Coto	Alajuela	4427867
124	Juan Coto	Alajuela	4427867	569	Eric Soto	Limón	7679899
149	Ana Salas	Heredia	2375643	124	Juan Coto	Alajuela	4427867
267	Eva Mora	Heredia	2379898	569	Eric Soto	Limón	7679899

Figura 3.9 Ejemplos de relaciones de conjuntos

Una operación conjuntista puede aplicarse en relaciones con atributos que tienen nombres diferentes, pero representan el mismo concepto. Es el caso de los atributos Nombre y Nombre-Estudiante en las relaciones originales del ejemplo anterior. En este caso, se debe especificar el nombre del atributo resultante.

Por otra parte, es importante mencionar que existe una variante del operador unión que se llama *unión externa* -del inglés *outer union*- y que se representará por el símbolo \oplus .

En la unión externa las relaciones involucradas no necesariamente deben ser del mismo grado y los atributos respectivos no necesariamente deben ser todos compatibles.

En este caso, en los atributos que solo aparecen una vez, los valores correspondientes serán valores nulos -representados por el símbolo @- junto con las tuplas de la otra relación. El resultado se da sobre la unión de los atributos de las dos relaciones.

Ejemplo. Sean las dos relaciones que aparecen en la figura 3.10.

NACIONALIDAD		OBRA	
Nombre	País	Nombre	Cuadro
Tamayo	Méjico	Vinci	La Gioconda
Quirós	Costa Rica	Manet	Olimpia
Goya	España	Amighetti	La Celestina
Manet	Francia	Picasso	Guernica

Figura 3.10 Ejemplos de dos relaciones

La unión externa de estas dos relaciones se puede representar según la figura 3.11.

NACIONALIDAD \oplus OBRA		
Nombre	País	Cuadro
Tamayo	Méjico	@
Quirós	Costa Rica	@
Goya	España	@
Manet	Francia	@
Vinci		La Gioconda
Manet		Olimpia
Amighetti		La Celestina
Picasso		Guernica

Figura 3.11 Ejemplo de la unión externa de dos relaciones

3.4.2 Operadores relacionales

Además de los operadores conjuntistas que se han estudiado, existen cuatro operadores adicionales que permitirán definir un lenguaje de consultas basado en esta álgebra. Estos operadores adicionales son: la proyección, la selección, el join y la división.

La Proyección

La *proyección* es un operador unario, es decir, se aplica sobre una sola relación a la vez y permite una descomposición de tipo vertical de la relación involucrada.

Sea $R(A_1, A_2, \dots, A_n)$ una relación, t una tupla de R y X un subconjunto de atributos de R .

En primer lugar, se define el concepto de tupla proyectada. La tupla resultante al eliminar los valores cuyos atributos no se encuentran en X , se llama *tupla proyectada* a X y se denota por

$$t[X]$$

Ejemplo. Sea la relación PINTOR(Nombre, País, Movimiento, Epoca) y $X = \{\text{Nombre, Epoca}\}$ un subconjunto de atributos de esta relación.

Si la tupla $t = [\text{Diego de Rivera, México, Precolombino, 1886-1957}]$, entonces la tupla t proyectada a X es la siguiente

$$t[X] = [\text{Diego de Rivera, 1886-1957}]$$

En segundo lugar, la proyección de la relación R sobre X se denota por

$$R[X]$$

y se define como el conjunto de tuplas de R proyectadas a X , es decir

$$R[X] = \{ t[X] / t \text{ es una tupla de } R \}$$

Ejemplo. En este caso, y en los siguientes ejemplos, los atributos se establecerán sobre un dominio de tipo de dato *ícono*.

En la figura 3.12, se puede apreciar la proyección de una relación $R(A,B,C)$ sobre $X = \{A,C\}$, es decir, $R[A,C]$.

R(A,B,C)			R[A,C]	
A	B	C	A	C
■	■	■	■	■

Figura 3.12 Ejemplo del operador proyección

Al efectuar una proyección, en la relación resultante pueden quedar tuplas duplicadas. Sin embargo, por la definición misma del modelo relacional, tales tuplas duplicadas se eliminan.

Con respecto a la representación gráfica de la proyección, ésta se presenta en la figura 3.13.

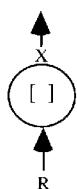


Figura 3.13 Representación gráfica de $R[X]$

Ejemplo. Se desea responder a la siguiente consulta sobre la base de datos mostrada en la figura 3.4.

Dar los nombres de los turistas del Club de Ecoturismo.

La respuesta a esta consulta sería $\text{RESULTADO} = \text{TURISTA} [\text{Nombre-Turista}]$ y el árbol asociado a dicha consulta se aprecia en la figura 3.14.

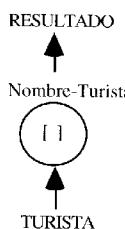


Figura 3.14 Árbol asociado a una consulta

La Selección

Por su parte, el operador selección, que también es un operador unario, brinda una descomposición horizontal de una relación. El resultado de una selección es el conjunto de tuplas que satisface una condición.

Así, sea $R(A_1, A_2, \dots, A_n)$ una relación y E una condición.

La selección de R bajo la condición E , se denota por

$$R(A_1, A_2, \dots, A_n) : E$$

y se define por

$$R(A_1, A_2, \dots, A_n) : E = \{ t \mid t \in R \wedge E(t) \text{ se verifica} \}.$$

Una condición de selección se define de la siguiente forma:

Si A y B son atributos de la relación R y a es un valor de un atributo, entonces

1. $A \Theta B$ y $A \Theta a$ son condiciones de selección, con $\Theta \in \{=, \neq, >, <, \geq, \leq\}$.
2. Si E_1 y E_2 son condiciones de selección, entonces también lo son:
 $E_1 \wedge E_2$
 $E_1 \vee E_2$
 $\neg E_1$
3. Cualquier condición de selección solo se puede generar como combinación de los dos puntos anteriores.

Debido a que los lenguajes de consultas relacionales que se pueden encontrar en el mercado se presentan en inglés, con respecto a los operadores lógicos, se utilizarán las siguientes equivalencias

AND por \wedge

OR por \vee

NOT por \neg

Ejemplo. En la figura 3.15, se muestra un ejemplo de una relación R(A,B,C) y una selección sobre el atributo B.

R(A,B,C)			R : (B = "↔")		
A	B	C	A	B	C
1	↔	Tree			
2	↔	Tree			
3	↔	Tree			
4	↔	Tree			

Figura 3.15 Ejemplo del operador selección

La representación gráfica de la selección se muestra en la figura 3.16.

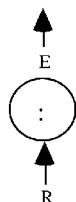


Figura 3.16 Representación gráfica de R : E

Ejemplo. Continuando con la base de datos de la figura 3.4, se desea responder a la siguiente consulta:

Dar información sobre los sitios que son volcanes y que se encuentran en África.

Entonces se puede aplicar el operador selección a la relación SITIO, de la siguiente forma:

RESULTADO = SITIO : (Tipo = 'Volcán' AND Continente = 'África').

En la figura 3.17 se representa el árbol correspondiente a esta consulta.

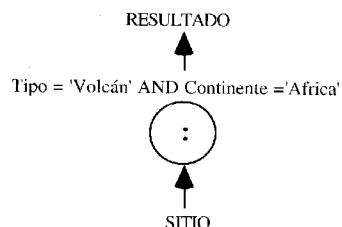


Figura 3.17 Árbol asociado a una consulta

El Θ - Join

El operador más importante del álgebra relacional es el llamado Θ -join o Θ -producto, cuyo papel primordial es fusionar, en una sola relación, dos relaciones que tengan al menos un atributo en común.

Formalmente, sea R una relación de grado n y S una relación de grado m, tal que existen atributos A y B en las relaciones R y S respectivamente que puedan ser comparados por medio de una operación Θ , en donde $\Theta \in \{<,>, \neq, =, \leq, \geq\}$.

Entonces, se define el Θ -join de R y S y se denota por $R \langle A \Theta B \rangle S$ el siguiente conjunto

$$R \langle A \Theta B \rangle S = \{ [t,s] / t \in R, s \in S \text{ y } t[A] \Theta s[B] \text{ se verifica} \}$$

Ejemplo. En la figura 3.18, se puede apreciar un ejemplo de un Θ -join entre las relaciones R(A,B) y S(B,C) y donde Θ es el operador " $=$ ".

R	S	$R \langle B=B \rangle S$				
A	B	B	C	A	B	C
stop	tree					
stop	tree					
stop	tree					
stop	tree					
elephant	tree					
elephant	tree					
elephant	tree					
elephant	tree					

Figura 3.18 Ejemplo del operador join

Cuando en un Θ -join, la operación Θ es la igualdad, algunos autores le llaman *equi-join*. En el caso de un equi-join, si solo se considera uno de los atributos comunes en la relación resultado se denomina *join natural*. En este libro simplemente se llamará *join*. El join entre dos relaciones R y S se denota por

$$R * S$$

Cuando se tienen dos relaciones las cuales tienen dos o más atributos comunes, la definición del join se puede generalizar.

Sea $R(X)$ y $S(Y)$ dos relaciones, con $Z = X \cap Y$ el conjunto de atributos comunes. Entonces, se define el join de R y S como el siguiente conjunto de tuplas

$$\{ [r|X-Z], s] / r \in R, s \in S \wedge r[Z] = s[Z] \}$$

Ejemplo. En la figura 3.19, se muestran dos relaciones R y S y el join correspondiente sobre dos atributos.

R			S		
A	B	C	B	C	D
a	1	x	1	x	m
b	2	y	4	y	n
c	3	z	3	t	ñ
c	3	w	3	z	l

$$R * S$$

A	B	C	D
a	1	x	m
c	3	z	l

Figura 3.19 Ejemplo de un join con dos atributos comunes

En la figura 3.20 se muestran las representaciones gráficas de los operadores Θ -join y join.

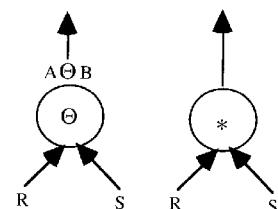


Figura 3.20 Representación gráfica del Θ -join y del join

Ejemplo. Sea la base de datos de la figura 3.4; se desea responder a la consulta:

Dar la información de los viajes y turistas.

Entonces, se puede aplicar un join a las relaciones TURISTA y VIAJE, es decir, $RESULTADO = SITIO * VIAJE$.

Por otra parte, si se tiene la siguiente consulta:

Dar la lista de las ciudades de salida utilizadas por los turistas cuyo número es superior a los de María y Carlos,

se debe utilizar un Θ -join de la siguiente forma:

$V1 = VIAJE[Número-Turista, Ciudad-Salida]$
 $T1 = (TURISTA : Nombre-Turista = 'María') [Número-Turista]$
 $T2 = (TURISTA : Nombre-Turista = 'Carlos') [Número-Turista]$
 $T3 = V1 \langle Número-Turista > Número-Turista \rangle T1$
 $T4 = T3 \langle Número-Turista > Número-Turista \rangle T2$
 $RESULTADO = T4[Ciudad-Salida].$

En la figura 3.21 se presenta el árbol asociado a la consulta planteada.

Este ejemplo sugiere varios comentarios. En primer lugar, cuando se busque la expresión que responda a una consulta, es recomendable reducir las diferentes relaciones a los atributos que son de salida y los

atributos involucrados en algún join que sea necesario en dicha consulta. Luego, sobre estas relaciones intermedias se aplican los operadores pertinentes.

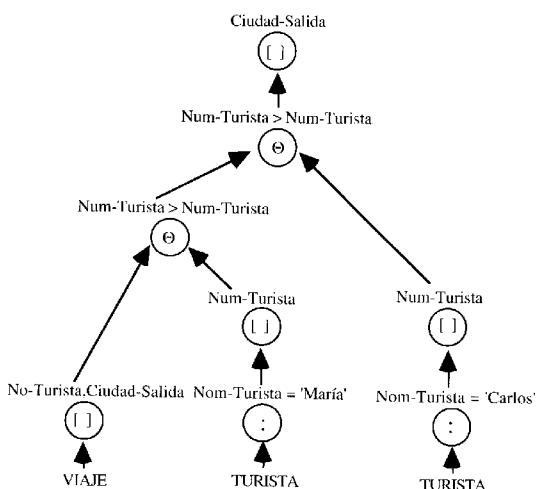


Figura 3.21 Árbol asociado a una consulta

Ejemplo. Se desea responder a la siguiente consulta:

Dar los nombres de los sitios y las fechas de llegada a estos sitios y que se encuentran en el continente asiático.

En este caso se involucran las relaciones SITIO y VIAJE por medio de un join. Previamente, se seleccionan las tuplas del continente asiático y luego se hace una proyección sobre los atributos Número-Sitio y Fecha-Llegada.

S1 = SITIO : (Continente = 'Asia')

S2 = S1[Número-Sitio, Nombre-Sitio]

V1 = VIAJE[Número-Sitio, Fecha-Llegada]

V2 = V1 * S2

RESULTADO = V2[Nombre-Sitio, Fecha-Llegada].

Al igual que el operador unión, se puede definir un operador *join externo* - del inglés *outer join* -. El join externo es una extensión del join y se define para rescatar las tuplas que quedan fuera en el caso de un join convencional. Existen situaciones en que es útil contar con dicha información.

En realidad, se debe hablar de tres tipos de operadores join externos y que formalmente se definen a continuación:

El *join externo izquierdo* entre dos relaciones R y S, se define como un join aumentado con las tuplas de R que no fueron involucradas en el join y para los atributos que no aplican, se llena con valores nulos

$$R * @ S = R * S \cup \{t / t = [r, @] \wedge r \in R\}$$

Por su parte, el *join externo derecho* entre dos relaciones R y S, se define como un join aumentado con las tuplas de S que no fueron involucradas en el join y para los atributos que no aplican, se llena con valores nulos

$$R @ * S = R * S \cup \{t / t = [@, s] \wedge s \in S\}$$

Finalmente, el *join externo completo* se define como una combinación de los dos join externos parciales, es decir, se define

$$R @* @ S = R * S \cup \{t / t = [r, @] \wedge r \in R\} \cup \{t / t = [@, s] \wedge s \in S\}$$

Ejemplo. Consideréense dos relaciones referentes a las nacionalidades y obras de pintores famosos

NACIONALIDAD

País	Nombre
Francia	Manet
Costa Rica	Quirós
Holanda	Rembrandt

OBRA

Nombre	Cuadro
Manet	Retrato Emile Zola
Rembrandt	La ronda de noche
Manet	Olimpia
Boticelli	La calumnia

Sobre estas relaciones se van a establecer los tres joins externos definidos anteriormente y se presentan en la figura 3.22.

NACIONALIDAD *@ OBRA

País	Nombre	Nombre	Cuadro
Francia	Manet	Manet	Retrato Emile Zola
Francia	Manet	Manet	Olimpia
Holanda	Rembrandt	Rembrandt	La ronda de noche
Costa Rica	Quirós	@	@

NACIONALIDAD @* OBRA

País	Nombre	Nombre	Cuadro
Francia	Manet	Manet	Retrato Emile Zola
Francia	Manet	Manet	Olimpia
Holanda	Rembrandt	Rembrandt	La ronda de noche
@	@	Boticelli	La calumnia

NACIONALIDAD @*@ OBRA

País	Nombre	Nombre	Cuadro
Francia	Manet	Manet	Retrato Emile Zola
Francia	Manet	Manet	Olimpia
Holanda	Rembrandt	Rembrandt	La ronda de noche
@	@	Boticelli	La calumnia
Costa Rica	Quirós	@	@

Figura 3.22 Ejemplo de joins externos

La División

A pesar de que algunos autores no consideran la división como un operador relacional, es importante considerarlo porque sirve para responder a una serie de consultas en donde implícitamente se involucra el cuantificador universal \forall y que se traduce en términos de expresiones como *para todos*, *para cada uno*, etc. Así, la división permite recuperar

CARLOS A. GONZALEZ A.

las tuplas de una relación que se encuentran asociadas con todas las tuplas de la otra relación.

Formalmente, sean $R(X)$ y $S(Y)$ dos relaciones, en donde $Y \subseteq X$. Entonces, la *división* de la relación R entre la relación S y denotada por

$$R \div S$$

se define como el conjunto de tuplas de R proyectadas a $X - Y$ tal que el producto cartesiano con S sea un subconjunto de R , es decir,

$$R \div S = \{ t \in R[X-Y] / \{t\} \times S \subseteq R \}$$

La división también se puede definir de la siguiente forma:

$$R \div S = \{ t / s \in S, \exists r \in R \text{ tal que } [t,s] = r \}$$

Ejemplo. En la figura 3.23, se tienen dos relaciones R y S , en donde R da una lista de personas que comen frutas y S las frutas disponibles. La división entre estas dos relaciones puede dar la lista de personas que comen todas las frutas.

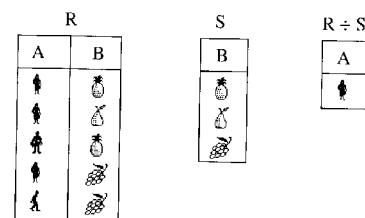


Figura 3.23 Ejemplo del operador división

En la figura 3.24 se muestra la representación gráfica de la división.

Ejemplo. Para responder a la consulta:

Dar los números de los turistas que han visitado todos los sitios de África, se requiere del operador división. En efecto, sean

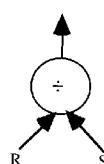


Figura 3.24 Representación gráfica del operador división

$R_1 = \text{VIAJE}[\text{Número-Turista}, \text{Número-Sitio}]$ y
 $R_2 = (\text{SITIO} : (\text{Continente} = \text{"Africa"})[\text{Número-Sitio}])$. Entonces
 $\text{RESULTADO} = R_1 \div R_2$.

El árbol asociado a esta consulta se presenta en la figura 3.25.

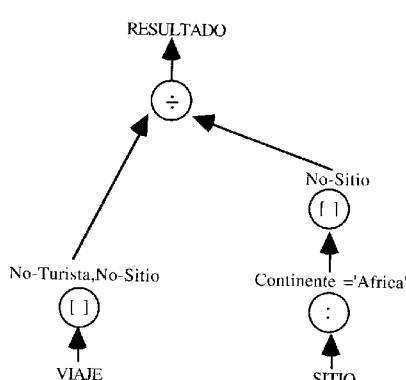


Figura 3.25 Árbol asociado a una consulta

Es interesante observar que algunos operadores pueden derivarse de otros. Por ejemplo, el join se puede definir a partir de los operadores producto cartesiano, selección y proyección, es decir :

$$R * S = (R \times S : (A_1 \Theta B_1)) [A_1, A_2, \dots, A_n, B_2, \dots, B_m]$$

Asimismo, la división de dos relaciones se puede representar por medio de la proyección y el producto cartesiano.

En efecto, sean $R(X)$ y $S(Y)$ dos relaciones, $Y \subseteq X$ y $Z = X - Y$. Entonces, $R \div S$ es equivalente a la siguiente expresión:

$$R \div S = R[Z] — (R[Z] \times S) — R[Z].$$

Sin embargo, dentro de los operadores relacionales se tienen algunos que no pueden ser derivados de otros, en cuyo caso se denominan operadores *primitivos*. Estos son: la unión, la diferencia, la proyección, la selección y el producto cartesiano.

Por otra parte, un lenguaje relacional se dice *completo* si por lo menos acepta los operadores primitivos.

Cuando se estudien los optimizadores de consultas, se va a profundizar en las equivalencias de expresiones basadas en los operadores algebraicos.

3.5 CALCULO DE PREDICADOS

Además del lenguaje del álgebra relacional, existe otro enfoque para expresar consultas llamado *cálculo de predicados*. En realidad, son dos enfoques: *cálculo de predicados de tuplas* y *cálculo de predicados de dominios*. Se puede mostrar que estos tres enfoques son de igual poder, en el sentido de que cualquier consulta hecha bajo uno de los enfoques, también se puede expresar en cualquiera de los otros dos restantes.

Antes de estudiar los lenguajes predicativos, es necesario introducir algunos conceptos de la lógica de predicados o lógica de primer orden, pues el cálculo de predicados de tuplas y el cálculo de predicados de dominios son simplificaciones y modificaciones de ésta.

En primer lugar, se introduce el concepto de predicado. Un *predicado*

$$P(x_1, x_2, \dots, x_n)$$

se define como el conjunto de valores (a_1, a_2, \dots, a_n) tales que la proposición $P(a_1, a_2, \dots, a_n)$ es verdadera.

Ejemplo. Sea el predicado P : *el estudiante de carné a, nombre n, que vive en la ciudad c, estudia en la institución i.*

Entonces $P(95-4567, Mario Coto, Cartago, ITCR)$ es verdadero.

Antes de definir el concepto de fórmula en la lógica de primer orden, se deben introducir los conceptos de término y de variable libre.

Un *término* se define recursivamente de la siguiente forma:

1. Toda constante es un término.
2. Toda variable es un término.
3. Si f es un símbolo de función n -aria y t_1, t_2, \dots, t_n son términos, entonces $f(t_1, t_2, \dots, t_n)$ es un término.
4. Todos los términos se generan solo usando las reglas anteriores.

Un *átomo* es una expresión de la forma $P(t_1, t_2, \dots, t_n)$ en donde P es un predicado y t_1, t_2, \dots, t_n son términos.

Una variable se dice *libre* en una expresión lógica si no se encuentra en el rango de un cuantificador.

Ejemplo. Sean las siguientes expresiones:

1. $(\exists y)(x > y)$. En este caso x es una variable libre y las dos ocurrencias de la variable y no lo son, es decir, y es una variable ligada.
2. $(\forall x)(\forall y)(\exists z)(Q(x,y) \Rightarrow R(z,u))$. En este caso u es la única variable libre de la expresión lógica.

Una *fórmula* en la lógica de primer orden, se define de la siguiente forma:

1. Un átomo es una fórmula.

2. Si F y G son fórmulas, entonces:

$\neg F, F \wedge G, F \vee G, F \Rightarrow G$ y $F \Leftrightarrow G$ son fórmulas.

3. Si F es una fórmula y x es una variable libre entonces: $(\forall x)F, (\exists x)F$ son fórmulas.

4. Cualquier fórmula se genera a partir de una aplicación finita de los pasos anteriores.

Si se desea profundizar en el tema de la lógica de predicados, se recomienda la excelente obra de Chang y Lee [CHAN73].

3.5.1 Cálculo de predicados de dominios

Una consulta que utiliza un lenguaje basado en el cálculo de predicados de dominio, que se llamará *predicativo de dominios*, se expresa de la forma:

$$\{ [x_1, \dots, x_n] / F(x_1, \dots, x_n) \}$$

En este caso F es una fórmula y cada x_i representa una variable de dominio, es decir, toma sus valores del dominio de un atributo.

De esta manera, la salida de una consulta que utiliza un lenguaje predicativo de variable dominio es un conjunto de tuplas $[a_1, \dots, a_n]$ tal que la expresión $F(a_1, \dots, a_n)$ es verdadera.

A continuación, se va a definir, en el contexto de los predicados de dominios, el concepto de fórmula. Antes, se introduce la noción de átomo.

Un *átomo* es una expresión de cualquiera de las tres formas siguientes:

1. $x_i \Theta x_j$, en donde los atributos que representan estas variables son comparables por medio de la operación Θ , con $\Theta \in \{ =, \neq, >, <, \geq, \leq \}$.

2. $x_i \Theta c$, en donde x_i es una variable de dominio y c es una constante.
3. $R(x_1, \dots, x_n)$, en donde R es una relación de grado n y x_1, \dots, x_n son variables o constantes.

Por su parte, una *fórmula* en el cálculo de predicados de dominios, se define de la siguiente forma:

1. Un átomo es una fórmula.
2. Si F y G son fórmulas, entonces:
 $\neg F, (F), F \wedge G, F \vee G, F \Rightarrow G$ y $F \Leftrightarrow G$ son fórmulas.
3. Si F es una fórmula y x es una variable libre, entonces:
 $(\forall x) F, (\exists x) F$ son fórmulas.
4. Cualquier fórmula se genera a partir de una aplicación finita de los pasos anteriores.

Ejemplo. Siguiendo con la base de datos de la figura 3.4, sobre el Club de Ecoturismo, se responderá a algunas consultas, utilizando el lenguaje predicativo de dominios.

1. *Dar los nombres de turistas del Club Ecológico.*
 $\{ [n] / \exists t \exists p \text{TURISTA}(t, n, p) \}$
2. *Dar información sobre los sitios que son volcanes y que se encuentran en África.*
 $\{ [s, n] / \text{SITIO}(s, n, 'Volcán', 'África') \}$
3. *Dar la información de los viajes y turistas*
 $\{ [t, n, p, v, s, fs, fl, cs] / \text{TURISTA}(t, n, p) \wedge \text{VIAJE}(v, t, s, fs, fl, cs) \}$
4. *Dar los nombres de los sitios y las fechas de llegada de estos sitios y que se encuentran en el continente asiático.*
 $\{ [n, fl] / \exists s \exists v \exists t \exists fs \exists fl \exists cs \text{SITIO}(s, n, 'Asia') \wedge \text{VIAJE}(v, t, s, fs, fl, cs) \}$.

✓ *Dar los números de los turistas que han visitado todos los sitios de África.*

$\{ [t] / \forall s \exists v \exists t \exists fs \exists fl \exists cs \exists n \text{VIAJE}(v, t, s, fs, fl, cs) \wedge \text{SITIO}(s, n, 'África') \}$.

A continuación, se introduce el QBE, que es un SABD relacional, cuyo lenguaje de consultas se sustenta sobre el enfoque del cálculo de predicados de dominios.

3.5.2 Lenguaje de consultas QBE

Las diferentes sesiones que se pueden hacer por medio de QBE inserción, modificación y supresión o borrado- se establecen sobre esquemas relacionales y que se presentan en pantalla como los armazones de las relaciones, según se introdujo al inicio del capítulo y se denominan *esqueletos*.

En efecto, una consulta en QBE, es un diagrama según se presenta en la figura 3.26.

Nombre de la relación	R	Atributo i	...	Atributo j
<i>Tipo de cláusula</i>				
Imprimir	P.			
Introducir	I.	a _i		a _j
Borrar	D.			
Modificar	U.			

Figura 3.26 Ejemplo tipo de una consulta en QBE

Ejemplo. Con el fin de comprender someramente el funcionamiento de QBE, se introducen las siguientes consultas:

1. *Introducir la tupla [148, Atacama, Desierto, América] en la relación SITIO*
2. *Suprimir el sitio de número 160 -llave primaria-*
3. *Reemplazar el tipo del sitio número 144, por Mar/Litoral*

sitio	num-sitio	nom-sitio	tipo	continente
L.	148	Atacama	Desierto	América
D.	160			
U.	144		Mar/Litoral	

4. Dar una lista con la información sobre los diferentes sitios

sitio	num-sitio	nom-sitio	tipo	continente
P.	_n	_m	_t	_c

En este caso, se aprecia que la consulta se expresa por medio de variables de dominio que son ejemplos, de ahí el término del lenguaje "query by example".

5. Dar la lista de los sitios que se encuentran en Africa

sitio	num-sitio	nom-sitio	tipo	continente
P.	_n	_m	_t	Africa

Utilizando el lenguaje predicativo de variable dominio, se responde de la siguiente forma $\{(s.n.t) / SITIO(s.n.t, 'Africa')\}$. Se puede apreciar la similitud de estos dos enfoques.

Es importante mencionar que si se desea la impresión solo de algunos atributos, entonces la cláusula de impresión P. puede trasladarse a las columnas de los atributos respectivos. Además, si las variables de dominio no se encuentran repetidas entonces se pueden suprimir. Así, si de la consulta anterior, solo se quisiera el nombre y tipos de sitio, se haría de la siguiente forma:

sitio	num-sitio	nom-sitio	tipo	continente
P.			P.	Africa

Por su parte, cuando en una consulta se involucran más de dos relaciones, las únicas variables de dominios que deben aparecer son las que traducen el join del álgebra relacional.

6. Dar los nombres de los volcanes visitados después del 5 de mayo de 1995 y en donde la ciudad de salida fue Salta

sitio	num-sitio	nom-sitio	tipo	continente
	_ns	P.		Africa

viaje	num-viaje	num-turista	num-sitio	f-salida	f-llegada	c-salida
			_ns	>05/05/95		Salta

Existe un esqueleto llamado CONDITIONS y se utiliza para expresar una o más condiciones, las cuales son difíciles de representar en las relaciones de base. Así, la siguiente consulta

1. Dar la lista de los números de turistas que han viajado y cuya ciudad de salida no es Londres

se puede responder usando CONDITIONS, de la siguiente forma

viaje	num-viaje	num-turista	num-sitio	f-salida	f-llegada	c-salida
P.						_x
conditions						
_x ≠ Londres						

Esta consulta también puede responderse de la siguiente forma

viaje	num-viaje	num-turista	num-sitio	f-salida	f-llegada	c-salida
			P._x	_x		Londres

El sistema QBE brinda asimismo funciones de cálculo y de ordenamiento:

- AVG calcula el promedio de un conjunto de valores
- MAX da el máximo de un conjunto de valores
- MIN brinda el mínimo de un conjunto de valores
- COUNT calcula la cardinalidad de un conjunto de valores
- SUM calcula la suma de un conjunto de valores
- G sirve para particionar o agrupar un conjunto de valores según los diferentes valores.

- **ALL** brinda la lista de todos los valores de un conjunto que satisfacen cierto criterio.

Ejemplo. Sean las siguientes consultas:

1. *Dar el número de viajes que salieron de Río*
2. *Dar el número de sitios de África*
3. *Agrupar los números de los viajes por ciudad de salida*

Cada una de estas consultas se expresan respectivamente en cada una de las siguientes líneas del armazón VIAJE.

viaje	num-viaje	num-turista	num-sitio	f-salida	f-llegada	c-salida
	P.		COUNT	Río		
			P.COUNT.ALL			G.

Se debe observar que la segunda expresión COUNT.ALL cuenta todos los valores aun si se encuentran repetidos. Así, es equivalente a poner P.COUNT en num-viaje, pues este atributo es una llave primaria.

3.5.3 Cálculo de predicados de tuplas

Otra alternativa de lenguaje relacional es el cálculo de predicados de tuplas.

Una consulta expresada bajo este concepto se expresa de la forma

$$\{ t / F \}$$

en donde t es una variable de tupla -una variable que recorre una relación- y F es una fórmula en donde la única variable libre es t .

Antes de definir el concepto de fórmula en el cálculo de predicados de tuplas, se introduce el de átomo.

Un **átomo** es una expresión de cualquiera de las tres formas siguientes:

1. $t_i[A] \Theta t_j[B]$ en donde $t_i[A]$ y $t_j[B]$ son variables de tuplas proyectadas a los atributos A y B , los cuales son comparables por medio de la operación Θ , con $\Theta \in \{ =, \neq, >, \leq \}$.
2. $t[A] \Theta c$, en donde $t[A]$ es una variable de tupla proyectada a A y c es una constante.
3. Una expresión de la forma $R \ni t$, que significa que t es una variable de tupla que recorre las tuplas de la relación R .

Ahora, una *fórmula* en los predicados de tuplas, se define de la siguiente forma:

1. Un átomo es una fórmula.
2. Si F y G son fórmulas, entonces:
 - $\neg F$, (F) , $F \wedge G$, $F \vee G$, $F \Rightarrow G$ y $F \Leftrightarrow G$ son fórmulas.
 - 3. Si F es una fórmula y x es una variable libre, entonces $(\forall x) F$, $(\exists x) F$ son fórmulas, con x variable libre en la fórmula F .
4. Cualquier fórmula se genera a partir de una aplicación finita de los pasos anteriores.

Ejemplo. Volviendo a la base de datos de la figura 3.4, sobre el Club de Ecoturismo, se van a responder algunas consultas, utilizando el lenguaje predicativo de tuplas.

1. *Dar los nombres de turistas del Club de Ecoturismo.*
 $\{ t [Nombre-Turista] / TURISTA \ni t \}$
2. *Dar información sobre los sitios que son volcanes y que se encuentran en el continente africano.*
 $\{ t / SITIO \ni t \wedge t[Tipo] = 'Volcán' \wedge t[Continente] = 'África' \}$
3. *Dar la información de los nombres de los turistas que hayan hecho un viaje antes del 1 de setiembre de 1995.*

{ t[Nombre-Turista] / (TURISTA \ni t \wedge \exists t₁ VIAJE \ni t₁ \wedge t[Número-Turista] = t₁[Número-Turista] \wedge t₁[Fecha-Salida] < 01/09/95 }.

4. Dar los nombres de los sitios y las fechas de llegada a estos sitios, que se encuentran en el continente asiático y que han sido visitados por turistas guatemaltecos.

{ t[Nombre-Sitio], t₁[Fecha-Llegada] / SITIO \ni t \wedge VIAJE \ni t₁ \wedge \exists t₂ TURISTA \ni t₂ \wedge t[Número-Sitio] = t₁[Número-Sitio] \wedge t[Continente] = 'Asia' \wedge t₁[Número-Turista] = t₂[Número-Turista]) \wedge t₂[País] = 'Guatemala' }

5. Dar los números de los turistas que han visitado todos los sitios de África.

{ t[Número-Turista] / VIAJE \ni t \wedge \forall s SITIO \ni s \wedge s[Continente] = 'África' \wedge s[Número-Sitio] = t[Número-Sitio] }.

3.5.4 Lenguaje de consultas QUEL

QUEL -QUEry Language-, es un lenguaje de consultas del sistema INGRES y es el lenguaje clásico que puede considerarse como del tipo predicativo de tuplas.

Una consulta básica en QUEL tiene la siguiente forma:

range of lista de variables tuplas	is nombre de relación
cláusula	lista de relaciones
where	expresión de selección

- La primera línea expresa el nombre de la relación resultado con las tuplas respectivas.

- En la segunda línea, la cláusula puede ser remplazada por

- **retrieve** búsqueda de datos
- **append to** inserción de datos
- **replace** modificación de datos
- **delete** borrado de datos.

- La tercera línea expresa la condición que deben satisfacer las tuplas que se desean recuperar.

Ejemplo. Responder a las siguientes consultas utilizando el lenguaje QUEL.

1. Introducir en SITIO, la tupla [148, Cerros Olga, Cerros/Montañas, Oceanía].
append to sitio (numero_sitio = 148, nombre_sitio = 'Cerros Olga', tipo = 'Cerros/Montañas', continente = 'Oceanía').

2. Dar los nombres de los sitios que son volcanes y que se encuentran en África.

```
range of s is sitio
retrieve (s.nom_sitio)
where s.tipo = 'Volcán' and s.continente = 'África'
```

Si se compara esta consulta a la que se puede hacer utilizando el lenguaje predicativo de tuplas introducido en la sección anterior, se puede observar la similitud.

3. Dur una lista de los nombres de los turistas y las ciudades de salida de aquellos que han viajado antes del 1 de marzo de 1996.

```
range of v is viaje
range of t is turista
retrieve into resultado(nom_turista, c_salida)
where v.num_turista = t.num_turista and fecha_salida < 01/03/96
```

Su equivalente en predicativo de tuplas es el siguiente:

{ t[Nombre-Turista], t[Ciudad-Salida] / TURISTA \ni t \wedge VIAJE \ni t \wedge t[Número-Turista] = t[Número-Turista] \wedge t[Fecha-Salida] < 01/03/96 }.

Si se desea cambiar el nombre de la relación resultado, se puede usar la variante

retrieve into nombre-relación ((A₁ = exp₁, ..., A_n = exp_n)

3.6 LOS 12 MANDAMIENTOS DE CODD

Una vez que el modelo relacional de Codd comenzó a establecerse como el modelo sobre el cual se deberían construir los nuevos SABDs,

muchos desarrolladores de software de bases de datos iniciaron una campaña llamando relacional a sus productos, muchos de éstos sin serlo.

Así, aparecieron SABDs que fueron concebidos sobre otros modelos de datos y se presentan posteriormente en el mercado con etiqueta relacional. Ante este caos de conceptos, Codd decide establecer un conjunto de reglas que permitan valorar en qué porcentaje un producto responde al calificativo de relacional. Estas 12 reglas, conocidas como los *12 mandamientos de Codd*, se basan en una regla fundamental, la cual dice:

Cualquier sistema que se diga un sistema de administración de bases de datos relacional -SABDR- debe administrar la base de datos relacional -BDR- solo a través de sus capacidades -consulta y actualización a nivel de la interfaz relacional-.

A continuación se presentan para cada una de las reglas una explicación, así como los beneficios que se generan. Se utiliza para ello, parte de la información presentada en [CODD85] y [SYBA88].

Regla 1	<i>Regla de la información</i> <i>Toda la información en una BDR -nombres de tablas, de columnas y de dominios- se representan explícitamente vía valores en tablas</i>
Explicación	El SABDR contará con un catálogo -a su vez es una BDR- que contendrá dicha información. El catálogo permite un mejor funcionamiento del SABD con respecto a derechos de acceso y recuperación después de fallas. Además, facilita el desarrollo de funciones -sistemas expertos, productos CASE- y el diálogo entre estas funciones y el SABD. Finalmente, permite al ABD un mantenimiento más eficiente de la base de datos.
Beneficio	Una visión única de los datos facilita el diseño y el aprendizaje. Así, solo se requiere conocer un lenguaje para accesar a todos los datos -a nivel del diseñador de la base de datos, del usuario y del ABD-.

Regla 2	<i>Regla del acceso garantizado</i> Cada valor de una tabla se accesa en forma lógica por medio de una combinación de un nombre de relación, un nombre de atributo y el valor de la llave primaria asociada.
Explicación	El acceso a los datos se puede hacer en diferentes formas. Sin embargo, se debe garantizar que al menos exista una que sea independiente de la especificación particular de la BDR. En esta situación, el concepto de llave primaria es fundamental.
Beneficio	No se requiere la utilización de punteros y direcciones y se da la independencia de los datos. Esto permite mayor eficiencia.
Regla 3	<i>Regla de los valores nulos</i> Los valores nulos son tomados en cuenta en un SABDR para representar la información faltante o no aplicable.
Explicación	Con el fin de establecer reglas de integridad de relación, el SABDR debe brindar mecanismos que permitan la prohibición de valores nulos en los atributos que conforman la llave primaria de una tabla, o en forma facultativa, los atributos que conforman una llave externa.
Beneficio	Con el uso de reglas que permiten el manejo de valores nulos, se pueden distinguir en las consultas y/o operaciones, aquellos resultados que involucran valores nulos, ceros o bien blancos.
Regla 4	<i>Regla del catálogo relacional</i> El catálogo de datos -descripción de la BDR- se debe representar a nivel lógico, de igual forma que las informaciones de la base de datos -bajo forma de relaciones-.
Explicación	Una de las consecuencias es que el usuario, ya sea programador de aplicaciones o usuario final, requiere manejar solamente un modelo de datos.
Beneficio	Al ABD se le facilita los cambios en el catálogo debido a que esto se puede hacer con las mismas cláusulas utilizadas para accesar cualquier otra tabla de la BDR.

muchos desarrolladores de software de bases de datos iniciaron una campaña llamando relacional a sus productos, muchos de éstos sin serlo.

Así, aparecieron SABDs que fueron concebidos sobre otros modelos de datos y se presentan posteriormente en el mercado con etiqueta relacional. Ante este caos de conceptos, Codd decide establecer un conjunto de reglas que permitan valorar en qué porcentaje un producto responde al calificativo de relacional. Estas 12 reglas, conocidas como los *12 mandamientos de Codd*, se basan en una regla fundamental, la cual dice:

Cualquier sistema que se diga un sistema de administración de bases de datos relacional -SABDR- debe administrar la base de datos relacional -BDR- solo a través de sus capacidades -consulta y actualización a nivel de la interfaz relacional-.

A continuación se presentan para cada una de las reglas una explicación, así como los beneficios que se generan. Se utiliza para ello, parte de la información presentada en [CODD85] y [SYBA88].

Regla 1	<i>Regla de la información</i> <i>Toda la información en una BDR -nombres de tablas, de columnas y de dominios- se representan explícitamente vía valores en tablas</i>
Explicación	El SABDR contará con un catálogo -a su vez es una BDR- que contendrá dicha información. El catálogo permite un mejor funcionamiento del SABD con respecto a derechos de acceso y recuperación después de fallas. Además, facilita el desarrollo de funciones -sistemas expertos, productos CASE- y el diálogo entre estas funciones y el SABD. Finalmente, permite al ABD un mantenimiento más eficiente de la base de datos.
Beneficio	Una visión única de los datos facilita el diseño y el aprendizaje. Así, solo se requiere conocer un lenguaje para accesar a todos los datos -a nivel del diseñador de la base de datos, del usuario y del ABD-.

Regla 2	<i>Regla del acceso garantizado</i> Cada valor de una tabla se accesa en forma lógica por medio de una combinación de un nombre de relación, un nombre de atributo y el valor de la llave primaria asociada.
Explicación	El acceso a los datos se puede hacer en diferentes formas. Sin embargo, se debe garantizar que al menos exista una que sea independiente de la especificación particular de la BDR. En esta situación, el concepto de llave primaria es fundamental.
Beneficio	No se requiere la utilización de punteros y direcciones y se da la independencia de los datos. Esto permite mayor eficiencia.
Regla 3	<i>Regla de los valores nulos</i> Los valores nulos son tomados en cuenta en un SABDR para representar la información faltante o no aplicable.
Explicación	Con el fin de establecer reglas de integridad de relación, el SABDR debe brindar mecanismos que permitan la prohibición de valores nulos en los atributos que conforman la llave primaria de una tabla, o en forma facultativa, los atributos que conforman una llave externa.
Beneficio	Con el uso de reglas que permiten el manejo de valores nulos, se pueden distinguir en las consultas y/o operaciones, aquellos resultados que involucran valores nulos, ceros o bien blancos.
Regla 4	<i>Regla del catálogo relacional</i> El catálogo de datos -descripción de la BDR- se debe representar a nivel lógico, de igual forma que las informaciones de la base de datos -bajo forma de relaciones-.
Explicación	Una de las consecuencias es que el usuario, ya sea programador de aplicaciones o usuario final, requiere manejar solamente un modelo de datos.
Beneficio	Al ABD se le facilita los cambios en el catálogo debido a que esto se puede hacer con las mismas cláusulas utilizadas para accesar cualquier otra tabla de la BDR.

Regla 5	<i>Regla de manipulación</i>
Explicación	Un SABDR comprende varios lenguajes de definición y de manipulación de datos.
Beneficio	Se mejora el rendimiento debido a que existe un enfoque que puede usarse para todas las operaciones de la BDR.

Regla 6	<i>Regla de actualización de vistas</i>
Explicación	Todas las vistas que pueden actualizarse son también modificables por el SABDR.
Beneficio	Se asegura la consistencia de los datos debido a que los cambios en las tablas de base se trasladan a la vista generada por dichas tablas.

Regla 7	<i>Regla de actualización de alto nivel</i>
Explicación	Una tabla -de base o derivada- puede servir de argumento no solamente a un operador de búsqueda de información sino también de actualización -inserción, supresión y modificación-.
Beneficio	Este requerimiento brinda al sistema la forma de determinar los caminos de acceso a los datos y obtener así el código más eficiente.

Regla 8	<i>Regla de la independencia física</i>
Explicación	Los programas de aplicación y las transacciones quedan inalterados a cualquier modificación concerniente a los métodos de acceso y de almacenamiento.
Beneficio	El SABDR debe establecer una clara diferenciación entre los aspectos físicos y semánticos por un lado y los aspectos físicos y de rendimiento del sistema por otro. Los programas de aplicación deben solo tratar con aspectos lógicos.

Regla 9	<i>Regla de la independencia lógica</i>
Explicación	Los programas de aplicación y las transacciones quedan inalterados a cualquier modificación -preservando la consistencia- efectuada en las relaciones de base.
Beneficio	Este aspecto se refiere al hecho de que el diseño de una base de datos puede cambiar en forma dinámica por razones de eficiencia.

Regla 10	<i>Regla de las restricciones de integridad</i>
Explicación	El SABD debe permitir la definición de reglas de integridad aplicativas -ligadas al universo real modelado- y almacenarse en el catálogo de la base de datos.
Beneficio	Además de las reglas de integridad de relación y referencial, otras reglas de integridad -reglas de empresa y regulaciones- se definen en términos de lenguajes de alto nivel y las definiciones se almacenan en el catálogo y no en los programas de aplicación.

Regla 11	<i>Regla de la independencia de distribución</i> Un SABD relacional es independiente de la ubicación física de los datos -BD centralizada o distribuida-.
Explicación	El SABD debe brindar un lenguaje que permita que los programas de aplicación y consultas ad-hoc queden intactos en forma lógica cuando se hacen cambios en la distribución física de los datos.
Beneficio	Se mejora la fiabilidad del sistema debido a que los programas de aplicación se ejecutarán incluso si los programas de aplicación y los datos son movidos a diferentes sitios.
Regla 12	<i>Regla de la seguridad de bajo nivel</i> Si el SABD posee un lenguaje de manipulación de datos de bajo nivel, este lenguaje no es accesible al usuario final.
Explicación	Si el SABDR posee un lenguaje que accesa a la base de datos por medio de un registro a la vez, este lenguaje no puede usarse para alterar las reglas de integridad. Así, el SABDR debe contar con un catálogo activo que contenga las reglas y debe poseer independencia lógica de datos.
Beneficio	Esto se requiere para la integridad de los datos.

EJERCICIOS y PREGUNTAS DE REPASO

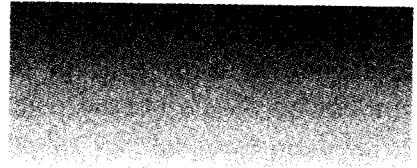
Considerar el siguiente esquema relacional sobre pintores y las épocas a las que pertenecieron, así como las obras que pintaron:

PINTOR(Nombre-Pintor, Fecha-Nacimiento, Nacionalidad)
 EPOCA(Nombre-Epoca, País-Origen, Duración)
 OBRA(Nombre-Obra, Nombre-Pintor, Nombre-Epoca)

Responder a las siguientes consultas utilizando el lenguaje algebraico, el predicativo de dominios y predicativo de tuplas:

- 3.1 Dar la información completa sobre todos los pintores.

- 3.1 Dar una lista de los pintores de nacionalidad peruana.
- 3.2 Dar una lista de los nombres de las obras que fueron pintadas por Chagall.
- 3.3 Establecer el pintor de la obra "Almuerzo en la hierba".
- 3.4 Dar una lista de los pintores que nacieron después de 1800 y los nombres de las obras.
- 3.5 Dar la lista de los nombres de los pintores, las épocas a las que pertenecieron, así como los países de origen de dichas épocas.
- 3.6 Dar una lista de todas las obras que pertenecen a la época impresionista.
- 3.7 Dar una lista de los nombres de pintores mexicanos de todas las épocas.
- 3.8 Dar una lista de los cuadros que no hayan sido pintados por pintores italianos.
- 3.9 Responder en QUEL y QBE a las siguientes consultas:
- 3.10 Introducir la tupla [Dalí, 1904, surrealista] en la relación PINTOR.
- 3.11 Suprimir de la relación OBRA todos los cuadros de Picasso.
- 3.12 Dar el número total de cuadros de Tamayo.
- 3.13 ¿Cuál es el pintor más joven de la relación PINTOR?
- 3.14 Agrupar los pintores por países.



4

Lenguaje de Consultas SQL

«El lenguaje no nos ha sido dado para comunicar nuestros sentimientos; nos damos cuenta de ello por el hecho de que todos los hombres sencillos sienten vergüenza de buscar palabras para sus emociones profundas; no las comunican más que por sus actos y se ruborizan de ver que los demás parecen adivinar sus motivos. Entre los poetas, a quienes generalmente la divinidad niega ese movimiento de pudor, los más nobles son monosilábicos en el lenguaje del sentimiento y dejan adivinar la sujeción, mientras que los verdaderos sacerdotes del sentimiento son con suma frecuencia insolentes en la vida práctica.»

Friedrich Nietzsche (1844-1900),
filósofo alemán.

4.1 INTRODUCCION

En el capítulo anterior se estudiaron los tres tipos de lenguajes **relacionales**, a saber: el lenguaje algebraico basado en los operadores **relacionales**, el lenguaje predicativo de tuplas y el lenguaje predicativo **de dominios**.

Los lenguajes de consultas de los diferentes productos relacionales que se encuentran en el mercado se basan por lo general en alguno de los **tres tipos** mencionados anteriormente, o alguna combinación de ellos.

Así, por ejemplo, el lenguaje de consultas *Perform* del SABD Informix, de la compañía Informix Software Inc., es un lenguaje de consultas basado en el lenguaje algebraico.

El SABD Ingres, desarrollado por M. Stonebraker y su grupo de la Universidad de Berkeley y comercializado desde 1981 por Relational Technology Inc. (RTI) posee el lenguaje de consultas *Quel* y se basa en el lenguaje predicativo de tuplas.

Asimismo, el lenguaje de consultas de QBE (Query By Example), producto desarrollado por M. M. Zloof de IBM en Yorktown Heights y comercializado desde 1980, es un lenguaje de consultas basado en el predicativo de dominios y se fundamenta en la presentación visual de cuadros, llamados *esqueletos*.

Sin embargo, lo que es interesante, dentro de este gran número de productos relacionales es la búsqueda de estándares por parte de investigadores, académicos y desarrolladores de SABDs. Esto, con el fin de poder trasladar bases de datos relacionales, que se han construido bajo un determinado producto relacional, a otros ambientes relacionales.

Uno de estos productos de estandarización, escogido por el grupo ANSI/SPARC, ha sido el lenguaje relacional *SQL*. Este lenguaje es un híbrido construido sobre la base del lenguaje algebraico y del predicativo de tuplas. Hoy en día, la gran mayoría de productos relacionales, incorporan el SQL como lenguaje de consultas.

El SQL tiene su origen en el System R, desarrollado por IBM Research Laboratory de San José, California, como parte de un programa de investigación sobre el modelo de datos relacional [BLAS76], [CHAM76].

Inicialmente, el SQL se llamó SEQUEL -Structured English as a Query Language- [CHAM76] y se basaba en el lenguaje SQUARE que utilizaba notaciones matemáticas e introducía el concepto de *bloque de selección/proyección*

```
select
from
where
```

Este bloque, también conocido como *bloque de calificación*, es el concepto fundamental del SQL y será estudiado con profundidad en la sección 4.3.

4.2 CONFIGURACION Y ENTORNO DEL SQL

4.2.1 Configuración del SQL

EL SQL es un lenguaje relacional de consultas, constituido de tres componentes principales:

- Un lenguaje de *control* de la base de datos
- Un lenguaje de *descripción* de datos
- Un lenguaje de *manipulación* de datos

4.2.2 Catálogo de la base de datos

El catálogo de cada base de datos es un conjunto de tablas, es decir, es a su vez una base de datos, que contiene información sobre las tablas, vistas, índices, así como derechos de acceso a la base de datos.

Entre la i tamaño, tipo conformació puede consul Por su parte, SABD.

A modo c catálogo del :

Nombre
syscolau
syscolum
syscomm
sysfkcons
sysindexe
syskeys
syspkcon
syssynon
systabaut
systabcon
systables
sysuserau
sysviews

4.2.3 Optimiza

El optimiza acceso a los da

Con el opti páginas que se memoria princ

o por IBM parte de un relacional English as a QUARE que *bloque de* *ación, es el* *didad en la* *uido de tres* *as, es decir,* *re las tablas,* *os.* Entre la información que contiene un catálogo se tiene el nombre, tamaño, tipo y valores válidos en cada columna, además de la conformación de cómo se acceden las tablas y las vistas. El catálogo puede consultarse como cualquier otra base de datos por medio del SQL. Por su parte, las actualizaciones del catálogo son realizadas por el propio SABD.

A modo de referencia, en la figura 4.1 se muestran las tablas del catálogo del SQLBase de Gupta.

<i>Nombre de la tabla</i>	<i>Contenido</i>
syscolauth	Privilegios de actualización en cada columna
syscolumns	Columnas de la base de datos
syscommands	Cláusulas almacenadas en la BD
sysfkconstraints	Reglas de llave externa en la BD
sysindexes	Indices de las tablas
syskeys	Columnas de los índices
syspkconstraints	Reglas de llave primaria en la BD
syssynonyms	Sinónimos de una tabla o vista
systabauth	Privilegios de cada tabla
systabconstraints	Reglas de integridad de las tablas en la BD
systables	Tablas o vistas de la BD
sysuserauth	Niveles de autorización de usuarios de la BD
sysviews	Vista en la BD

Figura 4.1 El catálogo del sistema SQLBase

4.2.3 Optimizador de consultas

El optimizador de consultas se encarga de escoger el mejor camino de acceso a los datos, solicitados por una consulta del tipo SQL.

Con el optimizador lo que se persigue es minimizar el número de páginas que se deben intercambiar entre la memoria secundaria y la memoria principal. En el capítulo 8 se profundiza en este tema.

4.2.4 Ejecución de una cláusula en SQL

La ejecución de una cláusula en SQL se cumple en varias etapas. En primer lugar, se tiene una fase de *análisis y verificación*, en donde se verifica si una cláusula se ha formulado en forma correcta. Posteriormente, la expresión se descompone en componentes más simples para que sean analizados por el optimizador. Además, se verifican los nombres de las tablas y columnas en el catálogo. En segundo lugar se tiene la etapa de *optimización*, en donde se usan estadísticas del almacenamiento de los datos por medio del catálogo. Así, se examinan varios posibles caminos de acceso a los datos, se calcula el costo de cada uno y se escoge el mejor. Finalmente, se genera un plan para la ejecución de la cláusula.

Antes de iniciar el estudio del lenguaje relacional SQL, y con el fin de mejorar la comprensión de las diferentes cláusulas que conforman este lenguaje mediante la presentación de ejemplos, se va a utilizar la base de datos del Club de Ecoturismo presentada en la figura 3.4 del capítulo anterior.

4.3 DESCRIPCION DE LA BASE DE DATOS

Para empezar, se puede decir que en el ambiente del lenguaje SQL una relación se denomina *tabla* -del inglés *table*- y un atributo se denomina *columna* - del inglés *column*-.

Con el lenguaje SQL se puede definir, suprimir y modificar bases de datos, tablas, índices y vistas.

Una vez que se ha establecido el esquema de aplicación, se procede a la implementación de la base de datos que representará la aplicación en cuestión.

Así, en primer lugar, para definir una nueva base de datos, se podría utilizar una cláusula como la siguiente -en SQL de SQLBase-:

```
create database nombre-base de datos ;
```

Ejemplo. Si se aplica dicha cláusula

En el estándar existe. Lo que se puede ver como datos general a la el fin de identifica

Sobre el nombre restricciones. Así Corporation, el n caracteres y el pr Informix-SQL bas 10 caracteres, etc

4.4 DEFINICIÓN

Una vez que se espacio en disco espacio donde se de datos.

La definición de diferenciarlas de siguiente cláusula

n varias etapas. En la creación, en donde se forma correcta. Los componentes más relevantes son el catálogo. Además, se crean en el catálogo. En la creación, en donde se usan el medio del catálogo. De acuerdo a los datos, se normalmente, se genera

SQL, y con el fin de las que conforman la base de datos, se va a utilizar la figura 3.4 del

del lenguaje SQL, y un atributo se

modificar bases de

lación, se procede a instalar la aplicación en

de datos, se podría SQLBase:-

Ejemplo. Si se desea crear la base de datos del Club de Ecoturismo, entonces se aplica dicha cláusula:

```
create database ecoturismo;
```

En el estándar SQL-92 [ANSI92], el concepto de base de datos no existe. Lo que existe en realidad es una sola base de datos y lo que se puede ver como una base de datos particular es la porción de la base de datos general a la cual se tiene acceso. Así, se introduce la cláusula con el fin de identificar la parte de la base de datos que pertenece a un usuario.

```
create schema nombre-base de datos  
authorization nombre -usuario;
```

Sobre el nombre de una base de datos, según el producto, existen restricciones. Así por ejemplo, si se usa el sistema SQLBase de Gupta Corporation, el nombre de la base de datos debe ser a lo sumo de ocho caracteres y el primer carácter debe ser alfabético. Por su parte, si se usa Informix-SQL bajo el sistema operativo UNIX, se pueden permitir hasta 10 caracteres, etc.

4.4 DEFINICION DE LA BASE DE DATOS

Una vez que se ha definido una base de datos, el SABD asigna un espacio en disco que será identificado como *ecoturismo* y será en este espacio donde se podrán definir las tres tablas que conforman dicha base de datos.

La definición de las tablas, y que serán llamadas *tablas de base* para diferenciarlas de las vistas o tablas derivadas, se hace por medio de la siguiente cláusula:

```
create table nombre-tabla  
(nombre-columna 1      tipo-dato 1,  
...  
nombre-columna n      tipo-dato n);
```

A modo de referencia, se usarán las características de algunos de los tipos de datos definidos en el SQL de SQLBase [SQLB94]:

- **char(n)** Una palabra de n caracteres, con $n \leq 254$.
- **varchar** Para almacenar palabras de más de 254 caracteres.
- **decimal(x,y)** Un número decimal de a lo sumo 15 dígitos, en donde y representa el número de dígitos decimales y x el número total de dígitos. El rango es de $-10^{15}+1$ hasta $10^{15}-1$.
- **integer** Un número entero entre -2147483648 y +2147483647.
- **smallint** Un número entero entre -32768 y +32767
- **date** Una fecha mm/dd/aa, en la secuencia de mes, día y año.

Por su parte, el nombre de la columna debe ser a lo sumo de 18 caracteres.

Con respecto al SQL-92, para que una tabla pertenezca a un esquema definido previamente, su definición sería de la siguiente forma:

```
create table nom-esquema.nombre-tabla
  (nombre-columna 1      tipo-dato 1,
   ...
   nombre-columna n      tipo-dato n);
```

Es importante mencionar que en una cláusula *create table*, se puede establecer que algunos de los atributos no acepten valores nulos. Asimismo, se pueden definir llaves primarias e integridad referencial por medio de la definición de llaves externas. Sin embargo, estos conceptos se introducen en la sección 4.6, dedicada a las reglas de integridad que se pueden manejar por medio del SQL.

Ejemplo. Si se desean crear las tablas que conforman la base de datos, se procede de la siguiente forma:

```
create table turista
  (número_turista      smallint,
```

nombre
país

```
create table
  (número_
  nombre_
  tipo_
  continen_
```

```
create table
  (numer_
  numero_
  numero_
  fecha_sa_
  fecha_ll_
  ciudad_s_
```

Una vez q
que se dese

- adicionar
- suprimir
- modificar

Para ello,
se muestra a

```

    nombre_turista      char(30),
    país                char(20));

create table sitio
    (número_sitio      smallint,
     nombre_sitio       char(30),
     tipo               char(20),
     continente        char(7));

create table viaje
    (numero_viaje      smallint,
     numero_turista    smallint,
     numero_sitio       smallint,
     fecha_salida      date,
     fecha_llegada     date,
     ciudad_salida     char(20));

```

Una vez que se han definido las tablas de la base de datos, es posible que se desee cambiar la estructura de algunas de ellas, es decir,

- adicionar una nueva columna,
- suprimir una columna existente, o
- modificar el tipo de datos de una columna existente.

Para ello, se utilizan respectivamente las cláusulas *alter table*, según se muestra a continuación:

alter table nombre-tabla add (nombre-columna 1 tipo-dato 1, ... nombre-columna n tipo-dato n);
alter table nombre-tabla drop (nombre-columna 1 , ... nombre-columna n);
alter table nombre-tabla modify (nombre-columna 1 nuevo tipo 1 ... nombre-columna n nuevo tipo n);

Ejemplo. Suponer que se desea adicionar la columna Estadía a la tabla VIAJE y que se debe suprimir la columna Fecha-Llegada. Entonces, se utiliza la cláusula alter table de la siguiente forma:

```
alter table viaje add (estadia smallint);
alter table viaje drop (fecha_llegada);
```

Por otra parte, si se desea suprimir una tabla de la base de datos, se utiliza la siguiente cláusula:

```
drop table nombre-tabla;
```

Una vez que se ha definido la base de datos, con sus tablas y columnas, se procede a la carga de las tuplas usando la cláusula *insert* como se aprecia a continuación:

```
insert into nombre-tabla
[ ( lista de columnas ) ]
values ( lista de valores );
```

Ejemplo. Si se desea cargar la tabla SITIO, se procede de la siguiente forma:

```
insert into sitio values (125, 'Isla Moorea', 'Mar/litoral', 'Oceanía');
insert into sitio values (126, 'Bahía Matsushima', 'Mar/litoral', 'Asia');
insert into sitio values (127, 'Irazú', 'Volcán', 'América');
insert into sitio values (128, 'Ngorongoro', 'Volcán', 'Africa');
insert into sitio values (129, 'Valle de la Muerte', 'Desierto', 'América');
insert into sitio values (130, 'Kilimandjaro', 'Volcán', 'Africa');
```

Una vez que en la tabla de base se han almacenado los datos, se puede desear modificar su contenido. Para ello, se utiliza la siguiente cláusula:

```
update nombre-tabla set
columna 1 = expresión 1,
columna 2 = expresión 2,
...
columna n = expresión n
[where ( condición de búsqueda )];
```

Ejemplo. Suponer que se desea modificar el continente en los sitios de África. Entonces se utiliza la cláusula update sitio where continente = 'África'.

Así, las nuevas tuplas serían:

```
(5125,'Isla Moorea','Mar/litoral','Oceanía');
(4126,'Bahía Matsushima','Mar/litoral','Asia');
(1127,'Irazú','Volcán','América');
(2128,'Ngorongoro','Volcán','Africa');
(1129,'Valle de la Muerte','Desierto','América');
(2130,'Kilimandjaro','Volcán','Africa');
```

Por su parte, para eliminar una tabla se utiliza la cláusula delete from.

Ejemplo. Suponer que se encuen-

trarán las siguientes tuplas en la tabla SITIO:

Una vez que se deseará borrar por medio de la cláusula delete from, se estudiará la manipulación de datos.

a la tabla VIAJE y que utiliza la cláusula alter

a base de datos, se

con sus tablas y
o la cláusula *insert*

a siguiente forma:

Oceanía');
ral', 'Asia');

):
('América');
r):

los datos, se puede
siguiente cláusula:

Ejemplo. Suponer que se deben incrementar los números de los sitios, según el continente en que se encuentran, de la siguiente forma: en 1000 los de América; en 2000 los de Africa, en 3000 los de Europa, en 4000 los de Asia y en 5000 los de Oceanía. Entonces dicha modificación se hace de la siguiente forma:

```
update sitio set numero_sito = numero_sito + 1000
where continente = 'América';
update sitio set numero_sito = numero_sito + 2000
where continente = 'Africa';
update sitio set numero_sito = numero_sito + 3000
where continente = 'Europa';
update sitio set numero_sito = numero_sito + 4000
where continente = 'Asia';
update sitio set numero_sito = numero_sito + 5000
where continente = 'Oceanía';
```

Así, las nuevas tuplas de la tabla SITIO son las siguientes:

```
(5125,'Isla Moorea','Mar/litoral','Oceanía');
(4126,'Bahía Matsushima','Mar/litoral','Asia');
(1127,'Irazú','Volcán','América');
(2128,'Ngorongoro','Volcán','Africa');
(1129,'Valle de la Muerte','Desierto','América');
(2130,'Kilimandjaro','Volcán','Africa');
```

Por su parte, si se desean suprimir algunas tuplas de una tabla se utiliza la cláusula siguiente:

```
delete from nombre-tabla
[where criterio de selección];
```

Ejemplo. Suponer que se deben suprimir las tuplas que corresponden a los sitios que se encuentran en Oceanía. Entonces, se hace lo siguiente:

```
delete from sitio where continente = 'Oceanía';
```

4.5 MANIPULACION DE LA BASE DE DATOS

Una vez que se ha definido la base de datos, se procede a consultarla, por medio de las cláusulas que permiten la manipulación de los datos y que se estudiarán a continuación.

4.5.1 Definición general de una consulta

La forma general de la extracción de datos de una base de datos, vía el lenguaje relacional SQL, puede establecerse de la siguiente forma:

- **select** lista de expresiones escalares del resultado -lista de atributos-.
- **from** lista de los nombres de las relaciones involucradas.
- **where** predicado de calificación sobre las tuplas-condiciones-.
- **group by** columna que sirve para particionar -lista de atributos-.
- **having** predicado de calificación sobre las particiones -condición-.
- **union** permite hacer la unión de dos bloques select.
- **order by** columna retenida de la lista del select para un ordenamiento.

A continuación se estudiarán estos conceptos.

La estructura fundamental del lenguaje SQL es el llamado bloque *SFW* o *bloque de selección/proyección*:

```
select      x1, x2,..., xn
from       r1, r2, ..., rm
where      e;
```

La cláusula *select x₁, x₂,..., x_n* traduce el operador de proyección del álgebra relacional y brinda la salida de la tabla resultado.

Por su parte, la cláusula *from r₁, r₂, ..., r_m* brinda la lista de las relaciones involucradas en la consulta.

Finalmente, *where e* es un criterio de selección, según se definió en el capítulo 3, que representa el predicado del operador selección del álgebra relacional.

Dentro
operadores

- de c
- de c
- dif
- de p
- de c
- las

A contin
equivalente

4.5.2 Oper

Los oper
modelo rel
cláusulas:

- **union**,
- **interse**
- **except**

Ejemplo :

C-AHOR

C-CORRIE

sobre los e

Se desea re

1. Dar los
c 25000 e

```
select  
from  
where
```

Dentro de una expresión *where* se pueden utilizar los siguientes operadores:

- de comparación: **=, <, >, <=, >=, <>**,
- de conjuntos: **union** -unión-, **intersect** -intersección- y **except** -diferencia-,
- de pertenencia de conjuntos: **in** y **not in**,
- de comparación de conjuntos: **contains** y **not contain** y
- las conectivas lógicas: **and, or** y **not**.

A continuación y mediante ejemplos, se van a estudiar los bloques equivalentes en SQL, de los diferentes operadores del álgebra relacional.

4.5.2 Operadores conjuntistas

Los operadores de conjuntos de unión, intersección y diferencia del modelo relacional se traducen respectivamente utilizando las siguientes cláusulas:

- **union**,
- **intersect**,
- **except**.

Ejemplo. Sean las relaciones

C-AHORRO(Número-cliente, Nombre-cliente, Dirección, Monto) y
C-CORRIENTE(Num-cliente, Nombre, Dirección, Saldo),

sobre los estados de las cuentas de ahorros y corrientes de los clientes de un banco.

Se desea responder, utilizando el SQL, a las siguientes consultas:

1. *Dar los números y las direcciones de aquellos clientes que tienen al menos ¢ 25000 en su cuenta de ahorros o al menos ¢ 75000 en su cuenta corriente.*

```
select      numero_cliente, dirección
from        cuenta_ahorro
where       monto > 25000
```

```

union
select num_cliente, dirección
from cuenta_corriente
where monto > 75000 ;

```

2. Dar los números de aquellos clientes que viven en Alajuela y que tienen un monto inferior de ¢ 200000 en su cuenta corriente.

```

(select numero_cliente
from cuenta_ahorro
where dirección = 'Alajuela')
intersect
(select num_cliente
from cuenta_corriente
where monto < 200000);

```

3. Dar toda la información referente a los clientes de San José o Heredia que solo tienen cuenta de ahorros.

```

(select *
from cuenta_ahorro
where dirección = 'San José' or dirección = 'Heredia')
except
(select *
from cuenta_corriente
where dirección = 'San José' or dirección = 'Heredia') ;

```

En vez de escribir como salida de una consulta todos los atributos de una tabla, el SQL permite el uso de un asterisco.

4.5.3 Proyección

La proyección de la relación R a los atributos X_1, X_2, \dots, X_n y que se traduce por $R[X_1, X_2, \dots, X_n]$, tiene el siguiente equivalente en SQL:

```

select x1, x2, ..., xn
from r ;

```

Ejemplo. C-

Dar la lista de los clientes que viven en Alajuela y que tienen una cuenta de ahorro.

La respuesta es:

```

select
from

```

Se puede ver que se desea eliminar la columna dirección de la consulta anterior. La forma:

```

select dirección
from

```

4.5.4 Selección

La selección es una operación que se realiza sobre una relación existente.

Ejemplo. Considerar la siguiente consulta:

Dar la lista de los diferentes tipos de sitios y de los continentes en donde se encuentran.

La respuesta a esta consulta se puede presentar de la siguiente forma:

```
select      tipo, continente  
from       sitio;
```

RESULTADO

Tipo	Continente
Mar/Litoral	Oceanía
Mar/Litoral	Asia
Volcán	América
Volcán	Africa
Desierto	América
Volcán	Africa

Se puede ver que esta consulta no elimina los valores duplicados. Si se desea eliminarlos, se debe utilizar la cláusula *distinct*, de la siguiente forma:

```
select distinct    tipo, continente  
from             sitio;
```

RESULTADO

Tipo	Continente
Mar/Litoral	Oceanía
Mar/Litoral	Asia
Volcán	América
Volcán	Africa
Desierto	América

4.5.4 Selección

La selección de las tuplas de la relación R que satisfacen la condición E y que se representa por R : E, posee el siguiente equivalente en SQL:

```
select      *
from       r
where      e;
```

Ejemplo. Sea la siguiente consulta:

Dar la lista de los miembros del Club de Ecoturismo que son costarricenses, panameños o jamaiquinos.

La respuesta a esta consulta es la siguiente:

```
select      *
from       turista
where      pais = 'Costa Rica' or pais = 'Panamá' or pais = 'Jamaica';
```

RESULTADO

Número-Turista	Nombre-Turista	País
300	Carlos	Costa Rica
302	John	Jamaica
303	Mario	Panamá

Ejemplo. Co

Dar toda la ..

Esta consulta

```
select
from
where
```

RESULTADO

No-Turista	Turista
301	Pier
301	Pier
302	John
303	Mari
304	Alí

Una segund
de un anidame
se puede apreci

Ejemplo. Dar
de agosto de 1•

```
select      nombre
from       sitio
where      numero
  (select      nombre
   from       sitio
   where      numero
```

El segundo s..
consulta, esto es

Por lo genera
de valores que e

4.5.5 Θ-Join

Considérense dos relaciones R y S y el join entre estas dos relaciones, es decir, $R \langle A \Theta B \rangle S$.

En SQL es posible representar esta situación, por medio de la siguiente expresión:

```
select      r.* , s.*
from       r, s
where      r.ai = s.bj ;
```

En este caso $R.A_i$ significa que A_i es una variable atributo que recorre R. Es evidente que en el criterio de selección se comparan los atributos que sirven para hacer el join.

Ejemplo. Considerar la siguiente consulta:

Dar toda la información sobre los turistas y sus viajes.

Esta consulta se resuelve así:

```
select    turista.* , viaje.*  
from      turista, viaje  
where     turista.numero_turista = viaje.numero_turista ;
```

RESULTADO

No-Turista	Nom-Turista	País	No-Viaje	No-Sitio	Fecha-Salida	Fecha-Llegada	Ciudad-Salida
301	Pierre	Francia	03-95	125	03/03/96	03/10/96	París
301	Pierre	Francia	05-95	128	07/04/96	07/14/96	Dar-es-Salam
302	John	Jamaica	07-95	128	07/05/96	07/12/96	Mombasa
303	Mario	Mario	04-95	129	07/06/96	07/14/96	Las Vegas
304	Alí	Túnez	06-95	127	07/28/96	08/13/96	San José

Una segunda forma de traducir un join entre dos tablas es por medio de un anidamiento de cláusulas *select*, con ayuda de la cláusula *in*, como se puede apreciar en el siguiente ejemplo.

Ejemplo. Dar la lista de los nombres de sitios que van a ser visitados antes del 1 de agosto de 1997.

```
select  nombre_sitio  
from   sitio  
where  numero_sitio in  
       (select  numero_sitio  
            from   viaje  
            where  fecha_salida < 08/01/97) ;
```

El segundo *select from where* es una expresión que se denomina *sub-consulta*, esto es, una expresión incorporada dentro de otra.

Por lo general, las sub-consultas se usan para representar el conjunto de valores que deben determinarse por medio de la cláusula *in*. En el

caso del ejemplo anterior, los números de sitios se seleccionan de aquellos que son determinados por la sub-consulta que involucra a la tabla VIAJE.

Un anidamiento no se reduce a una sub-consulta. En efecto, un anidamiento puede hacerse con más de dos sub-consultas, como se muestra en el siguiente ejemplo.

Ejemplo. Dar la lista de los nombres de turistas que han visitado al menos un volcán.

```
select nombre_turista
from turista
where numero_turista in
  (select numero_turista
  from viaje
  where numero_sitio in
    (select numero_sitio
    from sitio
    where tipo = 'Volcán')) ;
```

RESULTADO

Nombre-Turista
Pierre
John
Alí

Esta consulta anidada, en realidad lo que traduce es un join entre tres tablas. Una respuesta alterna a esta consulta es la siguiente:

```
select t.nombre_turista
from turista t, viaje v, sitio s
where t.numero_turista = v.numero_turista
and s.numero_sitio = v.numero_sitio
and s.tipo = 'Volcán';
```

Es importante mencionar que, cuando en la consulta anterior se escribe por ejemplo

turista t

significa que t es un TURISTA.

4.5.6 División

Con respecto a la división, se considera que lo que se busca es dividir las predicciones. Además, se considera que S(Y) y que su resultado es R.

Para entender el concepto de división, se estudiará el siguiente ejemplo.

Ejemplo. Sea la siguiente consulta:

Dar los números de los turistas que han visitado

Si se desea responder a esta pregunta, proceder de la siguiente forma:

SITIO1 =
VIAJE1 =
RESULTADO =

Por su parte, usarán la siguiente forma:

```
select numero_turista
from viaje
where (select numero_turista
       from sitio
       where tipo = 'Volcán') =  

      (select numero_turista
       from sitio
       where tipo = 'Volcán');
```

4.6 OTRAS CLAUSULAS

Para darle mayor flexibilidad, se incorporan otras cláusulas:

e seleccionan de
que involucra a la

a. En efecto, un
nsultas, como se

visitado al menos un

un join entre tres
iente:

nsulta anterior se

significa que t es una variable tupla que va a recorrer la tabla TURISTA.

4.5.6 División

Con respecto al operador división y su definición, se debe recordar que lo que se busca es traducir el cuantificador universal de la lógica de predicados. Además, que la división se hace sobre dos relaciones $R(X,Y)$ y $S(Y)$ y que su resultado es el siguiente:

$$R \div S = \{ t \in R[X-Y] / \{t\} \times S \subseteq R \}.$$

Para entender cómo se traduce el operador selección en SQL, se estudiará el siguiente ejemplo.

Ejemplo. Sea la siguiente consulta:

Dar los números de los turistas que viajan a todos los sitios

Si se desea responder a esta consulta mediante el lenguaje algebraico, se puede proceder de la siguiente forma:

SITIO1 = SITIO[Número-Sitio] (Divisor)
VIAJE1 = VIAJE[Número-Turista,Número-Sitio] (Dividendo)
RESULTADO = VIAJE1 \div SITIO1.

Por su parte, usando el SQL se podría responder a esta misma consulta de la siguiente forma:

```
select numero_turista
from viaje
where (select numero_sitio
       from viaje v
      where numero_turista = v.numero_turista)
      =
(select numero_sitio
   from sitio);
```

4.6 OTRAS CLAUSULAS DEL SQL

Para darle mayor poder de respuesta y eficiencia al lenguaje SQL, se incorporan otras cláusulas que permiten, por ejemplo, el ordenamiento

de una tabla, sacar promedios de columnas, mínimos, máximos, etc. Por supuesto, estas cláusulas son ajena a la definición del modelo relacional, pero son muy útiles dentro de los requerimientos de consulta que una organización pueda hacer a la base de datos.

Se van a estudiar algunas de estas cláusulas, utilizando para ello ejemplos.

4.6.1 Funciones de cálculo

El SQL brinda un conjunto de funciones de cálculo que permite obtener de un conjunto de valores, el máximo, el mínimo, el promedio, la cardinalidad y la suma de dichos valores. Estas funciones toman sus valores que dependen del subconjunto de filas regresadas por la cláusula *where* de un *select*. En la ausencia de un *where*, la función toma sus valores de todas las filas formadas por la cláusula *select*. A continuación se presentan dichas funciones.

- **count(*)** brinda el número de tuplas que satisfacen la cláusula *where*.
- **count(a)** brinda el número de valores diferentes de la columna *a*.
- **sum(a)** brinda la suma total de los valores en la columna *a*.
- **max(a)** brinda el mayor valor de la columna *a*.
- **min(a)** brinda el menor valor de la columna *a*.
- **avg(a)** brinda el valor promedio de los valores de la columna *a*.

Ejemplo. Dar el número de sitio más grande y el número de sitios diferentes que pueden visitarse.

```
select max(numero_sitio), count(numero_sitio)
from sitio;
```

El resultado de esta consulta aplicada a la base de datos definida en el capítulo anterior, se muestra a continuación

4.6.2 Group by

En algunas consultas se agrupan las tuplas de una relación en función de agrupamiento. Por ejemplo, si se tiene la tabla Ciudad, Monto) que indica la cantidad de población de cada ciudad en cada provincia en que se encuentra, se obtiene el valor del atributo Monto para cada provincia.

La cláusula *group by* agrupa las tuplas de una relación que están agrupadas por los atributos que establecen en la cláusula *group by*.

Ejemplo.

Para cada continente, dar el número de países y la población total de dicho continente.

select continente, count(pais), sum(monto)
from sitio
group by continente;

La tabla resultado es la siguiente:

4.6.3 Having

La cláusula *having* sirve para filtrar los resultados de una consulta en función que el resultado de la cláusula *group by* cumple con ciertas condiciones.

, máximos, etc. Por
ción del modelo
nientos de consulta

tilizando para ello

lculo que permite
mimo, el promedio,
ciones toman sus
das por la cláusula
 función toma sus
ct. A continuación

sfacen la cláusula

de la columna *a*.

a columna *a*.

s de la columna *a*.

e sitios diferentes que

definida en el capítulo

RESULTADO

130	6
-----	---

4.6.2 Group by

En algunas circunstancias es importante agrupar un conjunto de **tuplas** de una relación según los valores de algunas columnas -*columnas de agrupamiento*- . Por ejemplo, en el caso de la base de datos de un **banco**, si se tiene una tabla CLIENTE(Número-Cliente, Nombre, Ciudad, Monto) podría ser importante agrupar los clientes según la **provincia** en que viven. En este caso, se requiere agrupar las tuplas según **el valor del atributo Ciudad**.

La cláusula *group by* se usa para obtener conjuntos de tuplas **agrupadas** por los valores de las columnas de agrupamiento que se **establecen** en la cláusula.

Ejemplo.

Para cada continente, dar una lista del continente y número de sitios localizados en dicho continente.

```
select continente, count(*)
from sitio
group by continente;
```

La tabla resultado correspondiente se muestra a continuación

RESULTADO

Continente	Count(*)
Oceanía	1
Asia	1
América	2
Africa	2

4.6.3 Having

La cláusula *having* acompaña a la cláusula *group by* y tiene la misma **función** que el *where* en el *select from*, es decir, se usa para aplicar una o más condiciones de selección a los grupos de tuplas.

Ejemplo Sea la siguiente consulta:

Dar una lista de aquellos tipos de sitio, de los cuales existen al menos tres del mismo tipo.

```
select      tipo
from       sitio
group by   tipo
having     count(*) >= 3;
```

RESULTADO

Tipo
Volcán

Una de las características importantes en la cláusula *having* se refiere al hecho de que puede ser utilizada una función de cálculo, situación que no se da en una cláusula *where*.

4.6.4 Order by

La cláusula *order by* sirve para ordenar los resultados de una consulta según los valores de una o más columnas.

Ejemplo. *Dar la lista de los números de sitios y sus nombres organizados en orden alfabetico.*

```
select      numero-sitio, nombre_sitio
from       sitio
order by   nombre_sitio
```

SITIO

Número-Sitio	Nombre-Sitio
126	Bahía Matsushima
127	Irazú
125	Isla Moorea
130	Kilimandjaro
128	Ngorongoro
129	Valle de la Muerte

El *order by* se
deseo de tener u
palabras clave a
forma ascendente

4.6.5 Exist

Con respect
fórmula de pred
papel de tal cuan

Ejemplo. *Dar la*

```
select      nombre
from       turista
where      exist
(select    nombre
from     sitio
where    numero-sitio = t.numero-sitio
and     nombre_sitio = t.nombre_sitio)
and      nombre like '%a%'
```

Este resultado

de las consultas sig

```
select      t.nombre
from       turista t
where      t.numero-sitio
and      nombre like '%a%'
```

El *order by* se puede acompañar de las cláusulas *asc* y *desc* según el ~~se~~ de tener un orden ascendente o descendente. Si ninguna de las dos ~~o~~ clave aparece en el *order by*, el sistema ordena por defecto en ~~de~~ ascendente.

5 Exist

Con respecto al cuantificador existencial que puede aparecer en una fórmula de predicados, el SQL brinda la cláusula *exist* para traducir el papel de tal cuantificador.

Ejemplo. Dar la lista de los nombres de turistas que viajan al sitio 128.

```
select    nombre_turista
from      turista
where     exist
          (select    *
           from      viaje
           where     numero_turista = turista.numero_turista
           and       numero_sitio = 128);
```

RESULTADO

Nombre-Turista
John
Pierre

Este resultado también puede obtenerse por medio de alguna de las dos consultas siguientes:

```
select    t.nombre_turista
from      turista t, viaje v
where     t.numero_turista = v.numero_turista
          and v.numero_sitio = 128;

select    t.nombre_turista
from      turista t
```

```

where numero_turista in
(select numero_turista
from viaje v
where numero_turista = t.numero_turista
and numero_sitio = 128);

```

Por otra parte, la forma negativa del *exist* es *not exist* y sirve para responder a algunas consultas de una cierta complejidad.

Ejemplo. *Dar los nombres de los turistas que no han hecho un viaje durante el año.*

```

select nombre_turista
from turista t
where not exist
(select *
from viaje
where numero_turista = t.numero_turista );

```

RESULTADO

Nombre-Turista
Carlos

4.6.6 View

Con respecto al resultado de una consulta, que en el caso del SQL siempre es una tabla, ésta se puede hacer permanente dentro de la base de datos. En este caso se dice que esta tabla resultado es una *vista* o una *tabla derivada*.

Una vista, en efecto, es una tabla derivada pues su existencia está condicionada a la existencia de las tablas de base que entraron a participar en la expresión de la consulta solicitada.

Sin embargo, a pesar de que esta tabla no es de base, el usuario la puede utilizar en forma transparente para que pueda formar

parte de otras
siguiente:

Ejemplo. La sig

create view
as **select**
from
where
and

brinda una tab
respectivamente
nombre_sitio.

Cuando se es
la expresión· alg
modificación em
reflejarse en dich

Es importante
pero puede a vec
una serie de regla

- Una vista d
siempre y
primaria -o
- Una vista d
puede actua
- Una vista qu
cálculo no s

parte de otras consultas. La forma general de una vista es la siguiente:

```
create view nombre-vista  
[(nombre-col 1, ..., nombre-col n)]  
as select  
      from  
      where ;
```

Ejemplo. La siguiente vista:

```
create view lugar_visitado (#turista, nombre, lugar)  
as select t.numero_turista, t.nombre_turista, nombre_sitio)  
       from turista t, sitio s, viaje v  
      where t.numero_turista = v.numero_turista  
        and s.nombre_turista = v.nombre_turista ;
```

brinda una tabla cuyos atributos son *#turista*, *nombre* y *lugar* derivados respectivamente de los atributos *numero_turista*, *nombre_turista* y *nombre_sitio*.

Cuando se establece una vista, lo que se almacena en el catálogo es expresión algebraica que da origen a dicha vista. Así, cualquier modificación en las tablas de base involucradas en la vista podrá reflejarse en dicha vista.

Es importante mencionar que la actualización de una vista es posible, pero puede a veces ser muy confuso. Sin embargo, se pueden establecer una serie de reglas [ELM94].

- Una vista definida sobre una sola tabla de base se puede actualizar siempre y cuando los atributos de la vista contengan la llave primaria -o alguna llave alterna- de la relación de base.
- Una vista definida como el join de varias tablas de base, no se puede actualizar.
- Una vista que se define utilizando los agrupamientos o funciones de cálculo no se puede actualizar.

Finalmente, si se desea suprimir una vista, se utiliza la siguiente cláusula:

```
drop view nombre-vista ;
```

4.6.7 Create index

La cláusula *create index* crea un índice sobre una o más columnas de una tabla. Después de que se ejecuta tal cláusula, el SABD optimiza la recuperación de los datos pues éstos se pueden encontrar sin necesidad de recorrer la tabla en forma completa.

La forma de la cláusula es la siguiente

```
create index nombre-índice on nombre-tabla  
(nombre-columna 1,..., nombre columna n) ;
```

Es importante observar que el hecho de crear un índice para una tabla dada, debe estar condicionado a varias situaciones. Por ejemplo, si se tiene una tabla de varios cientos de miles de tuplas y las consultas que se hacen sobre esta tabla siempre recuperan un alto porcentaje de las tuplas, crear un índice bajo estas condiciones se convertiría en algo inútil. Estas situaciones, de decidir si se procede a crear o no índices, puede afectar o mejorar el rendimiento de una base de datos. Sin embargo, estas situaciones forman parte de lo que se llama *afinamiento* de la base de datos -*tuning* en inglés- y queda fuera del ámbito de este libro.

4.7 SQL INCORPORADO

El lenguaje SQL puede utilizarse al interior de un lenguaje de programación de alto nivel como Ada, Pascal, PL/I, Cobol, etc.

En la figura 4.2 se muestra un ejemplo de una consulta SQL incorporada dentro de un programa en PL/I.

Las cláusulas del SQL son reconocidas por un pre-compilador, gracias al prefijo **exec sql**.

En la cláu
nombre de u
y sirve para h
SQL, a un r
programació
conjunto.

Figura 4.2

Al definirse
posiciona un p

La cláusula
una llamada er
asociada al cu

Por su parte
por medio del
y desplazar el
recibida por el
cuando ya no h

vista, se utiliza la siguiente

vista ;

sobre una o más columnas de cláusula, el SABD optimiza la eden encontrar sin necesidad

nombre-tabla
re columna n) ;

crear un índice para una tabla situaciones. Por ejemplo, si se e tuplas y las consultas que se n alto porcentaje de las tuplas, convertiría en algo inútil. Estas r o no índices, puede afectar o e datos. Sin embargo, estas na *afinamiento* de la base de ámbito de este libro.

interior de un lenguaje de
cal. PL/1, Cobol, etc.

mplo de una consulta SQL
/1.

idas por un pre-compilador,

En la cláusula **exec sql c be select...**, *c* es un cursor. Un *cursor* es el nombre de un conjunto de tuplas -que son el resultado de una consulta- y sirve para hacer el traslado de un modo conjuntista como es el caso en SQL, a un modo procedimental como es el caso de un lenguaje de programación. Además, es un puntero que se posiciona en una tupla del conjunto.

```
exec sql begin declare section;
    declare fs date;
    declare cs integer;
exec sql end declare section;
...
exec sql let a be
    select fecha_salida : fs, numero_turista : cs;
    from viaje
    where ciudad_salida = 'Tegucigalpa'
...
exec sql open c;
do while 'c no sea el último'
begin
exec sql fetch c;
    cómputo usando la tupla de c
end;
exec sql close c;
```

Figura 4.2 Ejemplo de una consulta SQL incorporada en un programa en PL/1.

Al definirse un cursor se genera la ejecución de la consulta y posiciona un puntero en la primera tupla del resultado.

La cláusula **exec sql open c**, provoca que el pre-compilador genere una llamada en PL/1 a un procedimiento para evaluar la consulta en SQL asociada al cursor.

Por su parte, las tuplas se concretan en el programa en forma explícita por medio del **fetch** que tiene por misión dejar disponible la tupla actual y desplazar el puntero hacia la siguiente tupla. De esta forma la tupla recibida por el programa puede ser procesada en un ciclo que concluye cuando ya no hay más tuplas en el conjunto resultado.

4.8 REGLAS DE INTEGRIDAD

Cuando se describió el modelo relacional, se presentaron tres reglas de integridad: integridad de dominio, integridad de relación e integridad referencial.

El estándar SQL no soporta estas reglas en forma explícita. Sin embargo, algunos productos, sí lo permiten. En el siguiente ejemplo se van a estudiar estas reglas.

Ejemplo. Sean las entidades PROFESOR y ESTUDIANTE y una asociación (imparte) 1-N que traduce, para cada profesor, los estudiantes a los cuales él les imparte un curso.

Esto se traduce en el modelo relacional por medio de dos tablas:

```
PROFESOR(Núm_Prof, Nom_Prof, Código-Dep, Dirección, Salario, Título)
ESTUDIANTE(Carné, Nombre, Dirección, Teléfono, Cód-Carrera, Núm_Prof)
```

Para definir, por ejemplo, la tabla ESTUDIANTE y las reglas de integridad involucradas, algunas versiones de SQL, como Oracle o Sybase, lo hacen de la siguiente forma:

```
create table estudiante
(carne          smallint,
nombre         char(30) not null,
dirección      char(30),
teléfono       smallint
cod_carrera    smallint
primary key (carne),
foreign key (num_profesor),
references (profesor);
```

Otra forma de definir una regla de integridad es por medio de las llamadas operaciones de disparo -en inglés *triggering operations*- . Las operaciones de disparo son reglas que guardan la validez de introducir, suprimir y actualizar operaciones. Al analizar una operación de disparo debe considerarse lo siguiente:

- el evento que activa la operación de disparo -inserción, supresión, modificación-

- el objeto
- condición

Algunos productos como el SQL-SERVER

La forma general SQL como

CREATE
OR
FOREIGN
ALTER
IF

Ejemplo. Supongamos que los números de carné de los estudiantes. Eso es lo que se hace en la siguiente sentencia:

```
create trigger
on turista .
for delete
as
delete *
from viaje,
where viaje.
```

Además, en el control de acceso, se tienen los comandos **rollback**, etc., respectivos sobre las operaciones.

4.9 LIMITACIONES

Parte de esta sección se refiere a [DATE89], en el que se tratan consultas SQL,

onal, se presentaron tres reglas de integridad de relación e integridad

reglas en forma explícita. Sin embargo. En el siguiente ejemplo se

y ESTUDIANTE y una asociación entre los estudiantes a los cuales él les

edio de dos tablas:

(Prof., Dirección, Salario, Título)
(Teléfono, Cód-Carrera, Núm-Prof)

ANTE y las reglas de integridad no Oracle o Sybase, lo hacen de la

tegridad es por medio de las *-triggering operations-*. Las guardan la validez de introducir, insertar una operación de disparo

disparo -inserción, supresión,

- el objeto que se involucra en la operación -atributos y/o tablas-
- condición bajo la cual se activa la operación.

Algunos productos permiten la definición directa de un disparador, como el SQL-Server.

La forma general de una operación de disparo en algunas versiones del SQL como el SQL-Server es la siguiente:

```
create trigger nombre de la operación
on nombre-tabla
for {insert/ update / delete} ...
as consulta SQL
if update nombre-columna [and/ or update];
```

Ejemplo. Suponer que se desea suprimir tuplas de la tabla TURISTA de tal manera que los números de los turistas asociados en la tabla de VIAJE se supriman también. Eso es lo que se llama *una regla de supresión en cascada*. Usando el SQL se podría responder de la siguiente forma:

```
create trigger supresion_cascada
on turista
for delete
as
delete *
from viaje, deleted
where viaje.numero_turista = deleted.numero_turista;
```

Además, en el SQL existen cláusulas para garantizar la seguridad y el control de acceso concurrente, como el **grant**, **revoke**, **commit**, **rollback**, etc. Estos conceptos serán introducidos en los capítulos respectivos sobre seguridad e integridad y control de la concurrencia.

4.9 LIMITACIONES DEL SQL

Parte de esta sección se basa en una discusión planteada por Date en [DATE89], en donde se evidencian las limitaciones del lenguaje de consultas SQL, de las cuales algunas hoy en día persisten.

Es claro que el SQL es un lenguaje relacional de gran difusión, ya que la mayoría de los SABDs relacionales actuales lo incorporan. Sin embargo, es importante mencionar algunos hechos limitantes del mismo y que van en contra del modelo relacional original.

En primer lugar, el SQL no cuenta con propiedades de cerradura, lo que sí ofrece el álgebra relacional. En efecto, una combinación bien formada, de operadores y relaciones, da como resultado una relación. Así por ejemplo, la cláusula básica en SQL es **select from where** que traduce una expresión del álgebra relacional en un orden particular:

proyección → selección → producto → tabla 1 → tabla 2

Sin embargo, no es posible expresar una consulta en SQL con el siguiente orden:

selección → proyección → producto → tabla 1 → tabla 2

Otra situación es el caso de la inversión de varios operadores relativos. Es el caso de los operadores unión y proyección, en donde se tiene la siguiente equivalencia

$$(R \cup S)[X] = R[X] \cup S[X]$$

Sin embargo, esta equivalencia no se da en SQL. Por ejemplo, sean las dos relaciones TURISTA_CRC y TURISTA_BOL de los turistas costarricenses y bolivianos del Club. Si se desea una lista de los nombres de estos turistas, se podría estar tentado a usar la siguiente expresión en SQL

```
select nombre_turista  
from (turista_crc union turista_bol);
```

Sin embargo, esta expresión no es válida en SQL. La respuesta tendría que ser

(select nombre_turista
from turista_crc
union
(select nombre_turista
from turista_bol))

Por otra parte, las funciones de...

Por ejemplo...

Dar la lista...

Se estaría...

select numero_viaje
from viajes
where
count (número_viaje) > 1

Sin embargo, de la siguiente forma...

select numero_viaje
from viajes
where
group by
having count (número_viaje) > 1

En este caso, se transforma una...

También, se...

Dar el número de pasajeros salido de P...

se responde...

e gran difusión, ya que
es lo incorporan. Sin
s limitantes del mismo
1.

iedades de cerradura,
rto, una combinación
como resultado una
SQL es **select from**
relacional en un orden

bla 1 → tabla 2

nsulta en SQL con el

bla 1→ tabla 2

de varios operadores
proyección, en donde

QL. Por ejemplo, sean
_BOL de los turistas
na lista de los nombres
siguiente expresión en

L. La respuesta tendría

```
(select nombre_turista  
from turista_crc)  
union  
(select nombre_turista  
from turista_bol);
```

Por otra parte, se puede hablar de la falta de ortogonalidad en las funciones de cálculo: **count**, **max**, **min**, etc.

Por ejemplo, para responder a la consulta

Dar la lista de sitios visitados por más de 10 turistas

Se estaría tentado a responder de la siguiente forma

```
select num_sitio, count (num_turista)  
from viaje  
where  
count (numero_turista) >10.
```

Sin embargo, esta respuesta es incorrecta en SQL. Debe responderse
de la siguiente forma:

```
select num_sitio, count (num_turista)  
from viaje  
where  
group by num_sitio  
having count (numero_turista) >10.
```

En este caso, no se tiene interés en agrupar la cláusula **group by** que transforma una tabla en un conjunto de tablas.

También, si se desea responder a la consulta:

Dar el número de viajes que se han realizado en el año y que han salido de Puerto Príncipe ,

se responde de la siguiente forma:

```
select count(num_viaje)
from viaje
where ciudad_salida = 'Puerto Príncipe'
```

Quizás una forma más natural sería

```
count (select num_viaje)
from viaje
where ciudad_salida = 'Puerto Príncipe')
```

Finalmente, se pueden resumir varias deficiencias adicionales del lenguaje SQL.

- No está preparado para ser ampliado con el fin de manejar datos multimedia y objetos complejos.
- No existe un operador explícito de join. Así, según el tipo de consulta que se haga para responder a un join -anidamiento o from <lista de tablas> se tienen tiempos diferentes de respuesta.
- El anidamiento es limitado y no es generalizado como el caso del álgebra relacional.
- El SQL no es un lenguaje basado en una sola filosofía, más bien es un híbrido entre el álgebra relacional y el predicativo de tuplas.
- No se tiene un cuantificador universal explícito, como sí lo hay para el cuantificador existencial *exist*.
- No se tiene un concepto de dominio explícito.
- El problema de los valores nulos es que no se evidencia la falta de un valor debido a que no se aplica o porque es un valor desconocido. El manejo de los valores nulos, como en una función promedio, varía de un producto a otro.

EJERCICIOS Y PREGUNTAS DE REPASO

4.1 Demostrar que el lenguaje SQL es relationalmente completo.

- 4.2 Dar lo ¿cómo
- 4.3 Expliqu
- 4.4 ¿Qué e
- 4.5 Describ
- 4.6 Estable ejercici
- 4.7. Respon
- Sea la s
- CHOFER
- CAMIC
- ENVIO
- Responde
- 4.8 Dar una que tiene
- 4.9 Dar una antes de
- 4.10 Dar una de 1993 camión
- 4.11 Dar una
- 4.12 Dar una con tod
- 4.13 Dar una

- 4.2 Dar los diferentes componentes de una operación de disparo ¿cómo se puede implementar en SQL una tal operación?
- 4.3 Explique algunas de las limitaciones del SQL
- 4.4 ¿Qué es un lenguaje huésped, y para qué sirve esta posibilidad?
- 4.5 Describir los pasos del procesamiento de una consulta en SQL.
- 4.6 Establecer la definición de la base de datos que aparece en los ejercicios del capítulo 3, utilizando el SQL.
- 4.7. Responder a las consultas de los ejercicios del capítulo 3 en SQL.

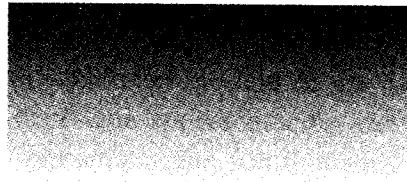
Sea la siguiente base de datos:

CHOFER(#Chofer, Nom-Chofer, Dirección, Fec-Inicio, Salario)
CAMION(#Camión, Marca, Tipo, Año, Capacidad)
ENVIO(#Chofer, #Camión, Fecha, Peso)

Responder en SQL a las siguientes consultas:

- 4.8 Dar una lista de los nombres de choferes que viven en San José y que tienen un salario superior de ₡ 100000.
- 4.9 Dar una lista de los camiones cisterna que hicieron un trayecto antes del 5 de octubre de 1995.
- 4.10 Dar una lista de los choferes que ingresaron después del 1 de enero de 1993 y que hicieron un envío superior de 5 toneladas en un camión Hino, en el año de 1994.
- 4.11 Dar una lista de los camiones agrupados por marca y año.
- 4.12 Dar una lista de los nombres de los choferes que han hecho envío con todos los camiones de la compañía.
- 4.13 Dar una lista del peso total de la mercadería transportada en 1995.

mente completo.



5

Proceso de Normalización

«Sin embargo, si descubrimos una teoría completa, debería ser comprensible a su debido tiempo, en líneas generales, para todo el mundo, no sólo para unos pocos científicos. Entonces todos, filósofos, científicos y gente de la calle, deberíamos poder tomar parte en la discusión de la pregunta de por qué existimos nosotros y el Universo. Si hallamos la respuesta a eso, será el triunfo definitivo de la razón humana..., porque entonces conoceremos la mente de Dios.»

Stephen Hawking (1942-),
físico británico.

5.1 INTRODUCCION

En el capítulo anterior se hizo una introducción del modelo de datos relacional. Este modelo, como se vio, hace una representación del mundo real por medio de un conjunto de estructuras que son llamadas relaciones, las cuales se constituyen de dominios y de atributos. Sin embargo, en la mayoría de los casos, estas relaciones iniciales tienen, dentro de su constitución, inconsistencias semánticas que provocan anomalías de manipulación. Estas anomalías de manipulación -pérdida de información, actualización redundante, inserción múltiple- pueden dejar la base de datos en un estado no deseado y no necesariamente va a reflejar los requerimientos de información de la organización.

Es por esta razón que, E. Codd [CODD70] propone un proceso llamado de *normalización de las relaciones* y que posteriormente fue ampliado por R. Fagin [FAGI77], C. Delobel [DELO78] y C. Zaniolo [ZANI81]. El proceso de normalización es un método propio del modelo relacional y consiste en descomponer las relaciones originales en otras más pequeñas con el fin de eliminar una serie de anomalías de almacenamiento y manipulación que se pueden dar en las relaciones iniciales y que conformarían la futura base de datos relacional.

Antes de introducir el proceso de normalización, se estudiará el siguiente ejemplo.

Ejemplo. Sea en un sistema hospitalario, lo referente a la atención de pacientes así como los médicos asignados. Esta situación se representa por medio de la relación que aparece en la figura 5.1.

Esta estructura tiene una serie de problemas relacionados con inconsistencias y anomalías de administración de las tuplas. Esto es, al insertar, modificar o suprimir tuplas dentro de esta relación, se van a presentar problemas.

En efecto, supóngase que se debe ingresar una nueva paciente de Limón llamada Astrid y que debe ser atendida en neurología. Entonces se le asigna el médico Mario. Esta situación obliga a la introducción de la tupla [3435678, Astrid, Limón, 125, Mario, Neurología, H. México].

De esta forma, se debe repetir la especialidad del médico así como el hospital en donde labora.

5.1 INTRODUCCION

En el capítulo anterior se hizo una introducción del modelo de datos relacional. Este modelo, como se vio, hace una representación del mundo real por medio de un conjunto de estructuras que son llamadas relaciones, las cuales se constituyen de dominios y de atributos. Sin embargo, en la mayoría de los casos, estas relaciones iniciales tienen, dentro de su constitución, inconsistencias semánticas que provocan anomalías de manipulación. Estas anomalías de manipulación -pérdida de información, actualización redundante, inserción múltiple- pueden dejar la base de datos en un estado no deseado y no necesariamente va a reflejar los requerimientos de información de la organización.

Es por esta razón que, E. Codd [CODD70] propone un proceso llamado de *normalización de las relaciones* y que posteriormente fue ampliado por R. Fagin [FAGI77], C. Delobel [DELO78] y C. Zaniolo [ZANI81]. El proceso de normalización es un método propio del modelo relacional y consiste en descomponer las relaciones originales en otras más pequeñas con el fin de eliminar una serie de anomalías de almacenamiento y manipulación que se pueden dar en las relaciones iniciales y que conformarían la futura base de datos relacional.

Antes de introducir el proceso de normalización, se estudiará el siguiente ejemplo.

Ejemplo. Sea en un sistema hospitalario, lo referente a la atención de pacientes así como los médicos asignados. Esta situación se representa por medio de la relación que aparece en la figura 5.1.

Esta estructura tiene una serie de problemas relacionados con inconsistencias y anomalías de administración de las tuplas. Esto es, al insertar, modificar o suprimir tuplas dentro de esta relación, se van a presentar problemas.

En efecto, supóngase que se debe ingresar una nueva paciente de Limón llamada Astrid y que debe ser atendida en neurología. Entonces se le asigna el médico Mario. Esta situación obliga a la introducción de la tupla [3435678, Astrid, Limón, 125, Mario, Neurología, H. México].

De esta forma, se debe repetir la especialidad del médico así como el hospital en donde labora.

ATENCION-DE-PACIENTES

Cédula	Nombre-Paciente	Dirección	Número-Médico	Nombre-Médico	Especialidad-Médico	Ubicación-Médico
2303776	Carlos	Alajuela	125	Mario	Neurología	H. México
3214568	María	Heredia	125	Mario	Neurología	H. México
4359087	Pedro	Alajuela	130	Carmen	Cardiología	San Rafael
1762344	Adrián	San José	135	Isabel	Hematología	Max Peralta
2764352	Juanita	San José	135	Isabel	Hematología	Max Peralta
6789114	Cecilia	Limón	135	Isabel	Hematología	Max Peralta

Figura 5.1 Ejemplo de relación no normalizada

Por otra parte, si el paciente Pedro es dado de alta y se borra la tupla correspondiente, se estaría perdiendo la información de que Carmen es una cardióloga y que trabaja en el Hospital San Rafael.

Finalmente, si Isabel es reemplazada por Erika, se deben hacer varias modificaciones en diferentes tuplas.

Para evitar este tipo de anomalías, se utiliza el proceso de normalización, que consiste en obtener una serie de relaciones, a partir de la relación original sin perder el contenido inicial, es decir, que si se aplica el operador join a las relaciones resultantes, se obtiene la relación original.

Así, una mejor solución al problema de la relación ATENCION-DE-PACIENTES, sería contar con dos relaciones, una con los datos referentes a los pacientes y otra con los datos referentes a los médicos, con un atributo común, que en este caso sería Número-Médico, según se aprecia en la figura 5.2.

PACIENTE

Cédula	Nombre-Paciente	Dirección	Número-Médico
2303776	Carlos	Alajuela	125
3214568	María	Heredia	125
4359087	Pedro	Alajuela	130
1762344	Adrián	San José	135
2764352	Juanita	San José	135
6789114	Cecilia	Limón	135

Fig. 5.2 continúa

En este caso, inserción en la 125].

Por su parte, si PACIENTE y Finalmente, si Médico. También es im conjunto de mé esto no era pos

El proceso de llamadas forma como los comp idea sería enton túnel y así o comportamiento el ejemplo anterior relaciones prese forma normal (5)

Pararelo a la definir varios c dependencia mul diferentes formas

Asimismo, se va a permitir la n

Especialidad	Ubicación-Médico
Oncología	H. México
Oncología	H. México
Oncología	San Rafael
Oncología	Max Peralta
Oncología	Max Peralta
Oncología	Max Peralta

y se borra la tupla que Carmen es una de las que deben hacer varias

a el proceso de relaciones, a partir es decir, que si se obtiene la relación

ON-DE-PACIENTES, a los pacientes y otra que en este caso sería

ero-
co

Fig. 5.2 continúa...

MEDICO

Número-Médico	Nombre-Médico	Especialidad	Ubicación-Médico
125	Mario	Neurología	H. México
130	Carmen	Cardiología	San Rafael
135	Isabel	Hematología	Max Peralta

Figura 5.2 Ejemplo de relaciones normalizadas

En este caso, si se desea introducir la paciente Astrid, solo debe hacerse una inserción en la relación PACIENTE, mediante la tupla [3435678, Astrid, Limón, 125].

Por su parte, si el paciente Pedro es dado de alta, se borra una tupla en la relación PACIENTE y no se pierde información de Carmen en la relación MEDICO. Finalmente, si Isabel es reemplazada, los cambios solo se hacen a nivel del Número-Médico.

También es importante notar que, bajo esta nueva circunstancia, se puede tener un conjunto de médicos, no todos asignados en un momento dado. En el caso anterior esto no era posible.

El proceso de normalización se compone de una serie de seis etapas llamadas *formas normales*. Si se pudiera uno imaginar estas etapas como los componentes de un túnel, según se aprecia en la figura 5.3, la idea sería entonces adentrarse lo más que se pueda al interior de este túnel y así obtener una serie de relaciones que tendrían un comportamiento exento de problemas como los que se mencionaron en el ejemplo anterior. Así, el objetivo de este proceso es llevar a que las relaciones presentes en la base de datos se encuentren todas en quinta forma normal (5FN).

Paralelo a la definición de las diferentes formas normales, es preciso definir varios conceptos, como son los de dependencia funcional, dependencia multivaluada y dependencia producto, que establecen estas diferentes formas normales.

Asimismo, se hará uso del llamado teorema de descomposición que va a permitir la navegación en las diferentes formas normales.

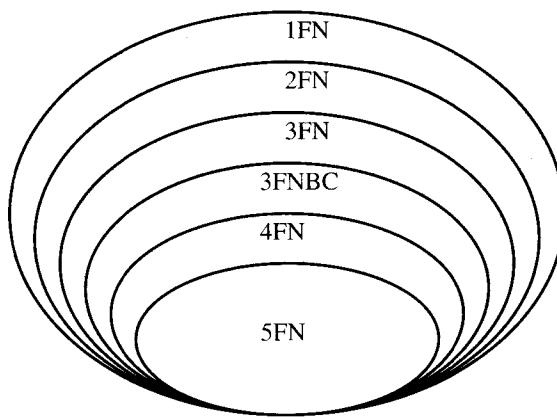


Figura 5.3 Proceso de normalización

Entre los beneficios que se pueden dar en una base de datos correctamente normalizada se encuentran los siguientes [DUTK89]:

- Reducir los problemas asociados con la supresión e inserción de tuplas.
- Reducir el tiempo asociado con modificaciones de las tuplas.
- Identificar problemas potenciales que pueden requerir un análisis adicional.
- Mejorar la información para la toma de decisiones referentes a la organización física de los datos.

5.2 PRIMERA FORMA NORMAL (1FN)

La *primera forma normal (1FN)* se refiere a la representación de una relación, en la cual los atributos son diferentes y los valores de cada uno de esos atributos son componentes atómicos. En este caso se tendría la definición formal de relación que se dio en el capítulo 3.

En los dos próximos ejemplos, se presentan dos relaciones que no se encuentran en primera forma normal.

Ejemplo. Se

EMPLEADO

Fecha-Pago3

y la relación
una organiza

EMPLEADO

Número	Fee
123	01/
134	01/
245	01/

Fig

A pesar de que representan una relación crear dos atribu Así, para trans considerar la si
EMPLEADO(N
y la relación re

Ejemplo. Sea la relación

EMPLEADO(Número, Nombre, Fecha-Pago1, Monto1, Fecha-Pago2, Monto2, Fecha-Pago3, Monto3)

y la relación de la figura 5.4, que representa el pago mensual de los empleados de una organización.

EMPLEADO

Número	Fecha-Pago1	Monto1	Fecha-Pago2	Monto2	Fecha-Pago3	Monto3
123	01/05/96	50000	01/06/96	55000	01/07/96	55000
134	01/05/96	70000	01/06/96	70000	01/07/96	70000
245	01/05/96	60000	01/06/96	60000	01/07/96	60000

Figura 5.4 Ejemplo de una relación que no se encuentra en 1FN

A pesar de que los atributos tienen nombres diferentes, en realidad las fechas de pago representan una sola fecha; lo mismo ocurre con los atributos montos. El problema de una relación como esta es el hecho que, por ejemplo, para cada nuevo mes se deben crear dos atributos. En este caso, la relación solo funciona para un trimestre. Así, para transformar una relación en una que se encuentre en 1FN, se podría considerar la siguiente relación:

EMPLEADO(Número, Fecha-Pago, Monto),

y la relación respectiva se puede apreciar en la figura 5.5.

Número	Fecha-Pago	Monto
123	01/05/96	50000
134	01/05/96	70000
245	01/05/96	60000
123	01/05/96	55000
134	01/05/96	70000
245	01/05/96	60000
123	01/05/96	55000
134	01/05/96	70000
245	01/07/96	60000

Figura 5.5 Ejemplo de relación en 1FN

Otro caso se puede apreciar en el ejemplo siguiente:

Ejemplo. Sea la relación de la figura 5.6.

TOUR

Número-Viaje	Sitio
06-96	(Irazú, Volcán)
04-96	(Valle de la Muerte, Desierto)
05-96	(Ngorongoro, Volcán)

Figura 5.6. Ejemplo de relación que no se encuentra en 1FN

En este caso, la relación no se encuentra en 1FN pues el atributo Sitio no es atómico.

Para trasladar la anterior relación a una que se encuentre en 1FN se puede proceder a la creación de dos atributos, Nombre-Sitio y Tipo-Sitio, según se aprecia en la figura 5.7, en lugar del atributo Sitio.

TOUR

Número-Viaje	Nombre-Sitio	Tipo-Sitio
06-96	Irazú	Volcán
04-96	Valle de la Muerte	Desierto
05-96	Ngorongoro	Volcán

Figura 5.7. Ejemplo de una relación en 1FN

5.3 DEPENDENCIAS FUNCIONALES

Cuando se introdujo el modelo de datos semántico se estudiaron las asociaciones entre entidades, es decir, entre conjuntos de objetos.

Asimismo, se pueden establecer asociaciones entre varios atributos de una misma relación, llamadas dependencias. Así, los valores de algunos atributos en una relación pueden determinar en forma única, el conocimiento de los valores de otros atributos de la misma relación.

Estas de
son muy co

5.3.1 Definición

En primera instancia

Formalmente, una de

atributos

Y, y se define

para cualquier

En este caso, se

Ejemplo. Se

CHOFER(

y la relación

CHOFER

Cédula

2303

1486

5456

4567

En este caso, se

- Para cada tupla, verifica si

uiente:

Desierto)
)

entra en 1FN

pues el atributo Sitio no es

re en 1FN se puede proceder
Sitio, según se aprecia en la

Tipo-Sitio
Volcán
Desierto
Volcán

en 1FN

nántico se estudiaron las
njuntos de objetos.

s entre varios atributos de
sí, los valores de algunos
nar en forma única, el
de la misma relación.

Estas dependencias traducen reglas semánticas que, como se sabe, son muy comunes en el modelaje del mundo real.

5.3.1 Definiciones básicas

En primer lugar, se define el concepto de dependencia funcional.

Formalmente, sea R un esquema de relación y sean X, Y subconjuntos de atributos de R . Se dice que existe una *dependencia funcional entre X y Y*, y se denota por

$$X \rightarrow Y$$

si para cualesquiera tuplas t_1 y t_2 de una relación R de este esquema tal que

$$t_1[X] = t_2[X] \text{ entonces } t_1[Y] = t_2[Y]$$

En este caso, a X se le llama el *determinante* y a Y el *dependiente*.

Ejemplo. Sea el esquema de relación

CHOFER(Cédula, Nombre, Dirección, Fecha-Ingreso, #Placa)

y la relación que se representa en la figura 5.8.

CHOFER

Cédula	Nombre	Dirección	Fecha-Ingreso	#Placa
2303776	Juan Mora	Alajuela	12/06/91	123456
1486756	Juan Mora	Heredia	10/12/87	103626
5456789	María Salas	Alajuela	10/12/87	123456
4567890	Carlos Mata	San José	12/11/86	67896

Figura 5.8 Ejemplo de relación

En este caso se pueden establecer varios hechos:

- Para cada número de cédula, existe un único nombre asociado, es decir, se verifica la dependencia funcional $\text{Cédula} \rightarrow \text{Nombre}$.

- Puesto que un camión puede ser conducido por varios choferes no se verifica la dependencia funcional $\#Placa \rightarrow \text{Cédula}$. En este caso se representa $\#Placa \rightarrow\!\!\! \rightarrow \text{Cédula}$.
- Se tiene $\text{Nombre} \rightarrow\!\!\! \rightarrow \#Placa$, puesto que dos personas distintas pueden tener el mismo nombre. Sin embargo, $\{\text{Nombre}, \text{Fecha-Ingreso}\} \rightarrow\!\!\! \rightarrow \#Placa$, se verifica, si se está seguro que dos personas con el mismo nombre no fueron contratadas el mismo día.

Es importante mencionar que la dependencia funcional es un concepto que deriva del significado de los datos y no del comportamiento de una relación dada. Así por ejemplo, en el esquema de relación CHOFER, se tiene que la dependencia funcional $\text{Fecha-Ingreso}, \text{Dirección} \rightarrow \text{Cédula}$ no se verifica debido a que dos personas diferentes pudieron haber ingresado el mismo día y vivir en la misma ciudad, aun cuando sí se verifica en la relación particular de la figura 5.8.

A partir de este concepto de dependencia funcional, se ha generado una serie de resultados teóricos que dan origen a lo que actualmente se conoce como *teoría relacional de bases de datos*.

En el siguiente apartado se va a hacer un recorrido rápido por esta teoría, estableciendo el concepto de cerradura irreducible de un conjunto de dependencias funcionales. Si se desea profundizar en el tema el lector puede consultar [ATZE93], [PARE89] y [MAIE83].

5.3.2 Cerraduras y coberturas mínimas

Una vez que se ha obtenido un conjunto de dependencias funcionales que se verifican en un esquema de relación, es interesante preguntarse si, a partir de este conjunto de dependencias se pueden inferir otras dependencias funcionales. El siguiente ejemplo introduce este concepto en forma intuitiva.

Ejemplo. Sea el siguiente esquema de relación:

CURSO(Grupo, Código, Profesor, Aula, Día, Hora)

y el siguiente conjunto F de dependencias funcionales:

$$F = \{ \begin{array}{l} f_1: \\ f_2: \\ f_3: \\ f_4: \\ f_5: \end{array}$$

A partir de...
Así, sea la...
y un profes...
imparte el...
funcional...
funcional...
informaci...
dice que f_6 ...
De igual m...
diferentes...
suprime a...
verifica, po...
Por otra pa...
 f_8 : Código,...
conforman...
 $F \Rightarrow f_8$.
Así, los a...
relación.

Según est...
de dependen...
pueden existi...
utilidad para

Sería inter...
un conjunto...
posibles depe...
dependencias...
cerradura, el...
necesario pre...

or varios choferes no se verifica la este caso se representa #Placa →

personas distintas pueden tener el ha-Ingreso} → #Placa, se verifica, smo nombre no fueron contratadas

ncional es un concepto que deriva iento de una relación dada. Así por tiene que la dependencia funcional ifica debido a que dos personas ia y vivir en la misma ciudad, aun la figura 5.8.

cia funcional, se ha generado igen a lo que actualmente se datos.

un recorrido rápido por esta irreducible de un conjunto profundizar en el tema el lector MAIE83].

de dependencias funcionales, es interesante preguntarse si, si se pueden inferir otras mplo introduce este concepto

ora)

ionales:

$$F = \{ \begin{array}{l} f_1: \text{Código, Profesor} \rightarrow \text{Aula, Día, Hora.} \\ f_2: \text{Código, Grupo} \rightarrow \text{Profesor} \\ f_3: \text{Código, Grupo} \rightarrow \text{Aula} \\ f_4: \text{Código, Grupo} \rightarrow \text{Día} \\ f_5: \text{Código, Grupo} \rightarrow \text{Hora} \end{array} \}$$

A partir de estas dependencias funcionales se pueden inferir otras nuevas.

Así, sea la dependencia funcional f_1 . Esto significa que dado un código de curso y un profesor se puede determinar el aula, el día y la hora en que el profesor imparte el curso. Pero, si además, se adiciona al determinante de esta dependencia funcional el atributo Grupo, con más razón se debe verificar la dependencia funcional $f_6: \text{Código, Profesor, Grupo} \rightarrow \text{Aula, Día, Hora}$, pues se tiene más información -tal vez innecesaria- para determinar el dependiente. En este caso, se dice que f_6 se infiere de f_1 y se denota $f_1 \Rightarrow f_6$.

De igual manera, se pueden inferir otras dependencias funcionales, utilizando otras diferentes reglas. Por ejemplo, sea la dependencia funcional f_1 , entonces si se suprimen atributos del dependiente, la dependencia funcional resultante también se verifica, por ejemplo, $f_7: \text{Código, Profesor} \rightarrow \text{Aula}$, entonces $f_1 \Rightarrow f_7$.

Por otra parte, se puede ver que la dependencia funcional

$f_8: \text{Código, Grupo} \rightarrow \text{Profesor, Aula, Día, Hora}$ se infiere de las dependencias que conforman el conjunto F . En este caso se dice que f_8 se infiere de F y se denota por $F \Rightarrow f_8$.

Así, los atributos Código, Grupo conforman una llave primaria para esta relación.

Según este ejemplo, se puede ver que un diseñador define una serie de dependencias funcionales que son evidentes para él. Sin embargo, pueden existir otras que quizás no consideró previamente y que son de utilidad para el diseño futuro de la base de datos.

Sería interesante contar con una herramienta que permita, a partir de un conjunto inicial de dependencias funcionales, establecer todas las posibles dependencias que se pueden inferir de este conjunto. Estas dependencias funcionales conforman un conjunto que se llama cerradura, el cual será definido en este apartado. Sin embargo, es necesario previamente introducir algunos conceptos.

La herramienta de inferencia está constituida por un conjunto de reglas -o axiomas-, las cuales se cumplen siempre, sin importar los atributos que se encuentren involucrados.

Este conjunto de reglas constituye un modelo que es *completo*. Esto significa que, si F es un conjunto de dependencias funcionales, cualquier dependencia funcional que se infiere a partir de F , se hace solo aplicando las reglas a las dependencias funcionales de F .

Este conjunto axiomático fue introducido por W. W. Armstrong [ARMS74] y se presenta a continuación.

Axiomas de inferencia de Armstrong

Sean X, Y, Z subconjuntos de atributos de una relación R , en donde se verifican las dependencias funcionales $X \rightarrow Y$ y

$Y \rightarrow Z$. Entonces, las siguientes reglas se cumplen:

A1 Reflexividad $X \rightarrow X$ se verifica siempre

A2 Aumento $X \rightarrow Y \Rightarrow X \cup Z \rightarrow Y$

A3 Transitividad $\{ X \rightarrow Y, Y \rightarrow Z \} \Rightarrow X \rightarrow Z$

A continuación se va a demostrar que estas reglas son siempre verdaderas.

A1: Sea X un conjunto de atributos. Sean dos tuplas t_1 y t_2 tales que $t_1[X] = t_2[X]$, entonces $t_1[X] = t_2[X]$, de donde se tiene que la dependencia funcional $X \rightarrow X$ se verifica.

A2: Suponer que la regla no se cumple, es decir que, existen dos tuplas t_1 y t_2 tal que $t_1[X \cup Z] = t_2[X \cup Z]$ y $t_1[Y] \neq t_2[Y]$. Puesto que $t_1[X \cup Z] = t_2[X \cup Z]$, entonces $t_1[X] = t_2[X]$. Ahora, como $t_1[X] = t_2[X]$ y $t_1[Y] \neq t_2[Y]$, esto implica que $X \rightarrow Y$ no se verifica, lo que contradice la hipótesis. De donde, la dependencia funcional $X \cup Z \rightarrow Y$ se debe verificar.

A3: Si :
y t_2
 \rightarrow
ent
dos
Así

A partir de
siguentes.

R. Entonces

A4 Unión

A5 Descor

A6 Pseud.

Para una c
[MAIE83].

Antes de i
Derivación de

En efecto,
U es el conjun

Se dice que
denota

si existen n de

1. $f_n = g$

2. $\forall i, i \in$
usando l

por un conjunto de
ore, sin importar los

ue es *completo*. Esto
uncionales, cualquier
e hace solo aplicando

r W. W. Armstrong

ong
a relación R, en
ales $X \rightarrow Y$ y
mplet:
empre
 $\rightarrow Y$
 $\Rightarrow X \rightarrow Z$

s reglas son siempre

tuplas t_1 y t_2 tales que
donde se tiene que la

decir que, existen dos
 $t_1[Y] \neq t_2[Y]$. Puesto
 $= t_2[X]$. Ahora, como
ea que $X \rightarrow Y$ no se
esis. De donde, la
e verificar.

A3: Si se verifica $X \rightarrow Y$, esto significa que si se tienen dos tuplas t_1 y t_2 tales que $t_1[X] = t_2[X]$, entonces $t_1[Y] = t_2[Y]$. Pero como $Y \rightarrow Z$ también se verifica, entonces dado que $t_1[Y] = t_2[Y]$, entonces $t_1[Z] = t_2[Z]$. De donde se tiene que, para cualesquiera dos tuplas t_1 y t_2 tales que $t_1[X] = t_2[X]$, entonces $t_1[Z] = t_2[Z]$. Así $X \rightarrow Z$ también se verifica.

A partir de este conjunto de reglas se pueden inferir otras como las siguientes. Sean X, Y, Z y W subconjuntos de atributos de una relación R . Entonces se verifican las siguientes reglas:

$$A4 \text{ } Unión \quad \{ X \rightarrow Y \text{ y } X \rightarrow Z \} \Rightarrow X \rightarrow Y \cup Z$$

$$A5 \text{ } Descomposición \quad X \rightarrow Y \Rightarrow X \rightarrow Z \text{ con } Z \subseteq Y$$

$$A6 \text{ } Pseudotransitividad \quad \{ X \rightarrow Y \text{ y } Y \cup Z \rightarrow W \} \Rightarrow X \cup Z \rightarrow W$$

Para una demostración completa de estas reglas, se puede consultar [MAIE83].

Antes de introducir el concepto de cerradura, se debe introducir el de derivación de dependencias funcionales.

En efecto, sea F un conjunto de dependencias funcionales, en donde U es el conjunto de atributos involucrados en F .

Se dice que la dependencia funcional $g: X \rightarrow Y$ se *deriva* de F y se denota

$$F \vdash g$$

si existen n dependencias funcionales f_1, f_2, \dots, f_n tal que

1. $f_n = g$
2. $\forall i, i \in \{1, \dots, n\}, f_i \in F$ o bien f_i se infiere de $\{f_1, f_2, \dots, f_{i-1}\}$ usando las reglas A1, A2 y A3.

Ejemplo. Sea F el siguiente conjunto de dependencias funcionales

$$\{ \begin{array}{l} f_1: \text{Dirección, } \#Depto \rightarrow \text{Fecha-Inicio, Salario} \\ f_2: \text{Nombre} \rightarrow \text{Dirección} \end{array} \}$$

Sea g: Nombre, #Depto \rightarrow Fecha-Inicio, Salario.

Entonces $F \vdash g$, como se muestra a continuación:

1. Nombre, #Depto \rightarrow Nombre, #Depto, por A1.
2. Nombre, #Depto \rightarrow Dirección, por A2 aplicada a f_2 .
3. Nombre, #Depto \rightarrow Nombre, #Depto, Dirección, por A3 aplicada a 1 y 2.
4. Nombre, #Depto, Dirección \rightarrow Fecha-Inicio, Salario, por A2 aplicada a f_1 .
5. g, por A3 aplicada a 3 y 4.

Una vez definido el concepto de derivación, es interesante establecer el conjunto de dependencias funcionales que se pueden derivar de un conjunto de dependencias funcionales F. A este conjunto se le llama *cerradura* de F y se denota por F^+ , es decir,

$$F^+ = \{f / F \vdash f\}$$

Por la definición de la cerradura, se puede ver que el conjunto de dependencias funcionales siempre es subconjunto de su cerradura, es decir, $F \subseteq F^+$. Esto se da porque cualquier dependencia funcional f de F, cumple que $F \vdash f$.

Además, la cerradura de una cerradura es la propia cerradura, es decir, $(F^+)^+ = F^+$.

Sea F un conjunto de dependencias funcionales. Se llama la *saturación* de un conjunto X con respecto a F, y se denota por X^+ , al conjunto de atributos que son determinados por las dependencias funcionales que se derivan de F, es decir,

$$X^+ = \{A / F \vdash X \rightarrow A\}$$

Utilizar
en resultado

Esto sig
puede der
equivalente
subconjunto

En efect
descomposi
 $A \in Y$, se e

Por otro
 $Y \in F^+$ por
que $X \subseteq X^+$
dependencia

Así, para
funcionales,
saturación de

A continu
un conjunto

Ejemplo. Se

{ Número,
Salario, N
Salario \rightarrow
Dirección
Fecha-Ing

Utilizando la saturación de un conjunto de atributos, se puede mostrar un resultado muy interesante y es el siguiente:

$$X \rightarrow Y \in F^+ \Leftrightarrow Y \subseteq X^+$$

Esto significa que, para determinar si una dependencia funcional se puede derivar de un conjunto F de dependencias funcionales, es equivalente mostrar que el dependiente de la dependencia funcional es subconjunto de la saturación del determinante.

En efecto, sea $X \rightarrow Y \in F^+$. Entonces, por la regla A5 -de descomposición-, las dependencias funcionales de la forma $X \rightarrow A$, con $A \in Y$, se encuentran en la cerradura de F .

Por otro lado, si $Y \subseteq X^+$, se tiene que la dependencia funcional $X^+ \rightarrow Y \in F^+$ por la regla A2 -de aumento-. Además, como $X \rightarrow X^+ \in F^+$ ya que $X \subseteq X^+$, se tiene que $X \rightarrow Y \in F^+$ por transitividad aplicada a las dependencias funcionales $X \rightarrow X^+$ y $X^+ \rightarrow Y$.

Así, para determinar la cerradura de un conjunto F de dependencias funcionales, es suficiente establecer un algoritmo que genere la saturación de un conjunto de atributos X .

A continuación se presenta un algoritmo que calcula la saturación de un conjunto X sobre un conjunto F de dependencias funcionales.

Ejemplo. Sea F el siguiente conjunto de dependencias funcionales

```
{ Número, Dirección → Salario
  Salario, Número → Teléfono
  Salario → Dirección
  Dirección → Teléfono
  Fecha-Ingreso, Dirección → Salario }
```

Ejemplo. Sea F el siguiente conjunto de dependencias funcionales

$$\{ \begin{array}{l} f_1: \text{Dirección, } \#Depto \rightarrow \text{Fecha-Inicio, Salario} \\ f_2: \text{Nombre} \rightarrow \text{Dirección} \end{array} \}$$

Sea g: Nombre, #Depto \rightarrow Fecha-Inicio, Salario.

Entonces $F \vdash g$, como se muestra a continuación:

1. Nombre, #Depto \rightarrow Nombre, #Depto, por A1.
2. Nombre, #Depto \rightarrow Dirección, por A2 aplicada a f_2 .
3. Nombre, #Depto \rightarrow Nombre, #Depto, Dirección, por A3 aplicada a 1 y 2.
4. Nombre, #Depto, Dirección \rightarrow Fecha-Inicio, Salario, por A2 aplicada a f_1 .
5. g, por A3 aplicada a 3 y 4.

Una vez definido el concepto de derivación, es interesante establecer el conjunto de dependencias funcionales que se pueden derivar de un conjunto de dependencias funcionales F. A este conjunto se le llama *cerradura* de F y se denota por F^+ , es decir,

$$F^+ = \{ f / F \vdash f \}$$

Por la definición de la cerradura, se puede ver que el conjunto de dependencias funcionales siempre es subconjunto de su cerradura, es decir, $F \subseteq F^+$. Esto se da porque cualquier dependencia funcional f de F cumple que $F \vdash f$.

Además, la cerradura de una cerradura es la propia cerradura, es decir, $(F^+)^+ = F^+$.

Sea F un conjunto de dependencias funcionales. Se llama la *saturación* de un conjunto X con respecto a F, y se denota por X^+ , al conjunto de atributos que son determinados por las dependencias funcionales que se derivan de F, es decir,

$$X^+ = \{ A / F \vdash X \rightarrow A \}$$

funcionales

A3 aplicada a 1 y 2.
por A2 aplicada a f_1 .

interesante establecer
pueden derivar de un
conjunto se le llama

er que el conjunto de
o de su cerradura, es
lencia funcional f de F ,

pia cerradura, es decir,

ionales. Se llama la
se denota por X^+ , al
por las dependencias

Utilizando la saturación de un conjunto de atributos, se puede mostrar un resultado muy interesante y es el siguiente:

$$X \rightarrow Y \in F^+ \Leftrightarrow Y \subseteq X^+$$

Esto significa que, para determinar si una dependencia funcional se puede derivar de un conjunto F de dependencias funcionales, es equivalente mostrar que el dependiente de la dependencia funcional es subconjunto de la saturación del determinante.

En efecto, sea $X \rightarrow Y \in F^+$. Entonces, por la regla A5 -de descomposición-, las dependencias funcionales de la forma $X \rightarrow A$, con $A \in Y$, se encuentran en la cerradura de F .

Por otro lado, si $Y \subseteq X^+$, se tiene que la dependencia funcional $X^+ \rightarrow Y \in F^+$ por la regla A2 -de aumento-. Además, como $X \rightarrow X^+ \in F^+$ ya que $X \subseteq X^+$, se tiene que $X \rightarrow Y \in F^+$ por transitividad aplicada a las dependencias funcionales $X \rightarrow X^+$ y $X^+ \rightarrow Y$.

Así, para determinar la cerradura de un conjunto F de dependencias funcionales, es suficiente establecer un algoritmo que genere la saturación de un conjunto de atributos X .

A continuación se presenta un algoritmo que calcula la saturación de un conjunto X sobre un conjunto F de dependencias funcionales.

Ejemplo. Sea F el siguiente conjunto de dependencias funcionales

```
{ Número, Dirección → Salario
    Salario, Número → Teléfono
    Salario → Dirección
    Dirección → Teléfono
    Fecha-Ingreso, Dirección → Salario }
```

Se desea encontrar $\{\text{Salario, Fecha-Ingreso}\}^+$. Entonces, si se aplica el algoritmo

Ciclo	XANT	XMAS	DF
	\emptyset	Salario, F-Ingreso	
1	Sal, F-ing	Sal, F-Ingr, Dir Sal, F-Ingr, Dir, Tel Sal, F-Ingr, Dir, Tel	Sal \rightarrow Dir Dir \rightarrow Tel F-Ingr, Dir \rightarrow Sal
2	Sal, F-ing, Dir, Tel	Sal, F-Ingr, Dir, Tel	

Así, se obtiene que $\text{XMAS} = \{\text{Salario, Fecha-Ingreso}\}^+ = \{\text{Salario, Fecha-Ingreso, Dirección, Teléfono}\}$.

Algoritmo de Saturación

Entrada: (F, X)

Salida: $X^+ (= \text{XMAS})$

```

begin
    XANT: =  $\emptyset$  ;
    XMAS:= X;
    while XANT  $\neq$  XMAS do begin
        XANT := XMAS
        for cada  $S \rightarrow T \in F$  do
            if  $S \subseteq$  XMAS then
                XMAS = XMAS  $\cup$  T
        end;
    return (XMAS)
end;
```

Sean F y G dos conjuntos de dependencias funcionales. Se dice que F y G son *equivalentes*, y se denota $F \equiv G$, si sus coberturas son iguales. Es decir,

$$F \equiv G \text{ si } F^+ = G^+$$

En este caso se dice que G es una *cobertura* de F .

ces. si se aplica el algoritmo

	DF
Sal	
Dir	Sal → Dir
Tel	Dir → Tel
F-Ing	Dir → Sal
	F-Ing, Dir → Sal

$F^+ = \{ \text{Salario, Fecha-} \}$

n
do
AS then
MAS \cup T

funcionales. Se dice que F coberturas son iguales, es

a de F.

Un conjunto de dependencias funcionales F se dice *no redundante*, si no existe un subconjunto propio G de F, tal que $F^+ = G^+$.

Sea F un conjunto de dependencias funcionales. Sea F^* el conjunto de las dependencias de la cerradura de F, en donde el determinado contiene un solo atributo, es decir, son dependencias funcionales *elementales*. Entonces $(F^*)^+ = F^+$.

Con esta definición, se puede definir el concepto de cobertura mínima.

Una cobertura G de F, se dice que es *mínima* si

1. Las dependencias funcionales de G son elementales.
2. No existe un subconjunto propio H de G tal que $H^+ = F^+$.

Para establecer un algoritmo que construya una cobertura mínima de un conjunto de dependencias funcionales, se debe en primer lugar, buscar las dependencias funcionales que son elementales, es decir, aquellas cuyos dependientes son atributos únicos. En segundo lugar, se busca eliminar las dependencias funcionales que son redundantes, es decir, eliminar aquellas $f \in F$, tal que $F - \{f\} \vdash f$.

5.3.3 Teorema de descomposición

A continuación, se presenta el teorema de descomposición que servirá como herramienta para desarrollar el proceso de normalización.

Teorema de Descomposición

Sea un esquema de relación $R(X, Y, Z)$, con X, Y y Z conjuntos de atributos de R, tal que la dependencia funcional $X \rightarrow Y$ se verifica en R. Entonces, la relación R se descompone en las relaciones $R_1 = R[X, Y]$ y $R_2 = R[X, Z]$, es decir, $R = R_1 * R_2$

Para una demostración de dicho teorema se puede consultar [PARE89].

Ejemplo. Considerar el esquema de relación

CURSO(Grupo, Código, Profesor, Aula, Día, Hora)

según se aprecia en la figura 5.9.

CURSO

Grupo	Código	Profesor	Aula	Día	Hora
01	C-1122	González	H-06	Lunes	09
03	C-2123	Helo	M-01	Lunes	14
04	C-3344	Helo	H-06	Martes	10
07	C-1515	Araya	H-06	Jueves	16

Figura 5.9 Ejemplo de relación

Según este esquema de relación se puede decir que para un profesor y un código dados, solo existe un aula, un día y una hora asociadas. En este caso, se puede decir que se establece la dependencia funcional

Código, Profesor → Aula, Día, Hora.

Si se aplica el teorema de descomposición usando esta dependencia funcional, se obtienen las siguientes relaciones:

$R_1 = R[\text{Código, Profesor, Aula, Día, Hora}]$ y

$R_2 = R[\text{Grupo, Código, Profesor}]$

con las relaciones que se muestran en la figura 5.10.

R_1

Grupo	Código	Aula	Día	Hora
01	C-1122	H-06	Lunes	09
03	C-2123	M-01	Lunes	14
04	C-3344	H-06	Martes	10
07	C-1515	H-06	Jueves	16

R_2

Grupo	Código	Profesor
01	C-1122	González
03	C-2123	Helo
04	C-3344	Helo
07	C-1515	Araya

Figura 5.10. Ejemplo de descomposición de relaciones

Es interesante notar que la descomposición engendra redundancia. En efecto, si la dependencia funcional $X \rightarrow Y$ se verifica, el conjunto de

atributos que componen Y no deben aparecer en la descomposición en las diferentes relaciones.

Resumiendo, la regla de integración que determina un único

Utilizando el concepto de una llave de una relación.

Sea $R(U)$ una relación de U . Se dice que

a. La dependencia

b. Ningún subconjunto

funcionalmente

5.4 SEGUNDA FASE

Antes de introducir la fase de optimización, es necesario conocer lo que significa la normalización de una relación. Se define la normalización de una relación R con respecto a un atributo A dependiente de A . Si A es un subconjunto funcionalmente dependiente de A , se dice que R es una relación en la forma normal de 2^{FN} .

Así, una relación en la forma normal de 2^{FN} , si se encuentra parcialmente de la forma normal de 2^{FN} .

Ejemplo. Considerar la relación

ITINERARIO(Número-Viaje → Destino, Fecha, Hora)

Puesto que cada viaje tiene una fecha y hora, la dependencia

Número-Viaje → Fecha y Hora, se aprecia en la figura 5.11.

ema se puede consultar

Día	Hora
Lunes	09
Lunes	14
Martes	10
Jueves	16

ue para un profesor y un código
das. En este caso, se puede decir

o esta dependencia funcional, se

10.

32

Grupo	Código	Profesor
01	C-1122	González
03	C-2123	Helo
04	C-3344	Helo
07	C-1515	Araya

ón de relaciones

n engendra redundancia. En
se verifica, el conjunto de

atributos que conforman X, se duplican en las dos relaciones que surgen de la descomposición. Es el único tipo de redundancia que será tolerada en las diferentes etapas del proceso de normalización y se le denomina *emigración de atributos*.

Resumiendo, se puede ver que una dependencia funcional es una regla de integridad que significa: *el conocimiento de un valor X determina un único valor de Y*.

Utilizando el concepto de dependencia funcional, se puede definir una *llave* de una relación de la siguiente forma:

Sea $R(U)$ un esquema de relación y sea X un subconjunto de atributos de U. Se dice que X es una llave de $R(U)$, si

- a. La dependencia funcional $X \rightarrow U$ se verifica en R y,
- b. Ningún subconjunto propio de X puede determinar funcionalmente a U.



Siblicofeca

5.4 SEGUNDA FORMA NORMAL (2FN)

Antes de introducir el concepto de segunda forma normal, se requiere conocer lo que significa dependencia parcial. En efecto, sea un esquema de relación R , X una llave para R y A un atributo no llave. Se dice que el atributo A depende *parcialmente* de X si se verifica $Y \rightarrow A$, en donde Y es un subconjunto propio de la llave X.

Así, una relación R se dice que se encuentra en *segunda forma normal (2FN)*, si se encuentra en 1FN y si ningún atributo no llave depende parcialmente de la llave primaria.

Ejemplo. Considerar la siguiente relación:

ITINERARIO(Número-Viaje, Fecha-Salida, Tarifa, Número-Sitio, Número-Turista).

Puesto que cada viaje tiene asociado una sola tarifa, en esta relación se verifica la siguiente dependencia funcional:

Número-Viaje \rightarrow Tarifa,

según se aprecia en la figura 5.11.

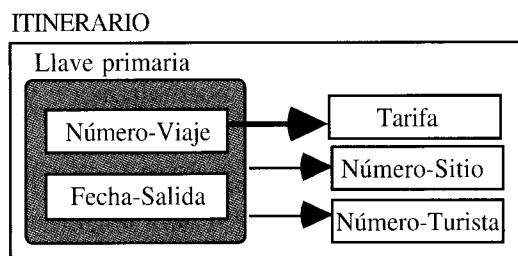


Figura 5.11 Ejemplo de una relación que no está en 2FN

En este caso, la relación ITINERARIO no se encuentra en 2FN ya que el atributo Tarifa depende parcialmente de la llave primaria que es la combinación de los atributos Número-Viaje, Fecha-Salida.

Debido a este hecho, se presentan una serie de anomalías de manipulación de datos. En efecto, si se desea introducir un valor de Número-Viaje, Tarifa se puede hacer solo si existe un valor correspondiente para la llave primaria Número-Viaje, Fecha-Salida.

Por otra parte, si se suprime un valor de la llave se puede perder un valor único de Número-Viaje, Tarifa.

Finalmente, si se desea modificar un valor de la Tarifa asociado a un valor de un Número-Viaje se puede tener inconsistencias o un alto costo de actualización para el caso de los valores duplicados.

Con el fin de eliminar estos problemas se aplica el teorema de descomposición a la dependencia funcional que viola la 2FN, en este caso $\text{Número-Viaje} \rightarrow \text{Tarifa}$. Al aplicar dicho teorema, se obtienen las relaciones siguientes:

ITINERARIO1(Número-Viaje, Fecha-Salida, Número-Sitio, Número-Turista) y COSTO-ITINERARIO(Número-Viaje, Tarifa),

según se aprecia en la figura 5.12.

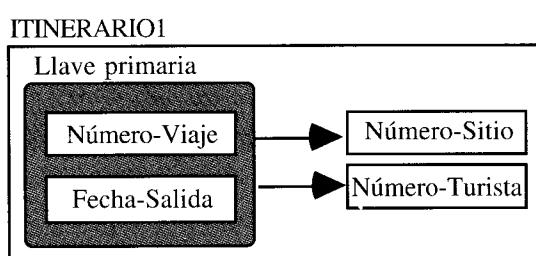


Fig. 5.12 continúa

Como se resuelven con persiste, como

5.5 TERCERA NORMALIZACIÓN

Antes de ir a definir previamen

Sea $R(X, Y)$ un subconjunto de R que cumple la 2FN y sea Z un subconjunto de X si existe

- se verifica
- no se verifica
- se verifica

Se dice que R cumple la 3FN si se establece la independencia funcional entre atributos

Ejemplo. Considere

ITINERARIO1

Se va a suponer que la siguiente

Número-Turista

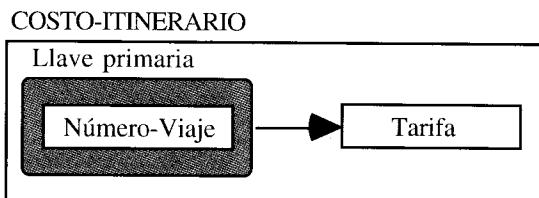


Figura 5.12 Ejemplo de relaciones en 2 FN

Como se puede observar, las anomalías antes mencionadas se resuelven con esta descomposición, sin embargo, otro tipo de problemas persiste, como se verá en la siguiente sección.

5.5 TERCERA FORMA NORMAL (3FN)

Antes de introducir el concepto de tercera forma normal, es preciso definir previamente, el término de dependencia transitiva.

Sea $R(X,Y,Z)$ un esquema de relación, en donde X , Y y Z son subconjuntos de atributos. Se dice que Z es *transitivamente dependiente* de X si existe Y tal que se dan las siguientes condiciones:

- se verifica $X \rightarrow Y$,
- no se verifica $Y \rightarrow X$,
- se verifica $Y \rightarrow Z$.

Se dice que una relación R se encuentra en *tercera forma normal* (3FN) si se encuentra en 2FN y no existe una dependencia transitiva entre atributos no llave.

Ejemplo. Considerar la relación siguiente:

ITINERARIO1(Número-Viaje, Fecha-Salida, Número-Sitio, Número-Turista)

Se va a suponer que cada turista visita un solo sitio durante el año. Esto significa que la siguiente dependencia funcional se verifica en la relación ITINERARIO1:

$\text{Número-Turista} \rightarrow \text{Número-Sitio}$.

Fig. 5.12 continúa...

Así el atributo Número-Sitio depende transitivamente de la llave y por lo tanto no se encuentra en 3FN, según se aprecia en la figura 5.13.

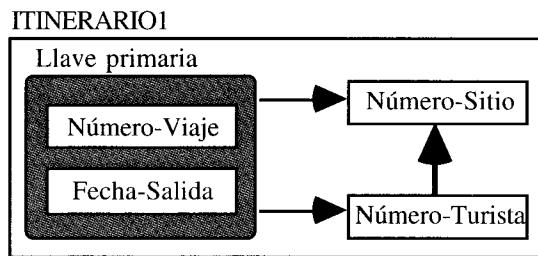


Figura 5.13 Ejemplo de relación que no está en 3FN

Al no encontrarse la relación ITINERARIO1 en tercera forma normal, se presentan varios problemas.

En efecto, si por ejemplo, se desea introducir un valor de Número-Turista, Número-Sitio, solo se puede hacer si existe un valor asociado de Número-Viaje, Fecha-Salida.

Por su parte, si se suprime un valor de Número-Viaje, Fecha-Salida se puede perder un valor único de Número-Turista, Número-Sitio.

Asimismo, si se desea modificar un valor Número-Turista, Número-Sitio, se puede tener un alto costo de actualización debido a la redundancia presente.

Para resolver el problema, se debe aplicar el teorema de descomposición a la dependencia funcional transitiva para obtener las dos relaciones

ITINERARIO2(Número-Viaje,Fecha-Salida,Número-Turista)
SITIO-VISITADO(Número-Turista,Número-Sitio),

según se aprecian en la figura 5.14.

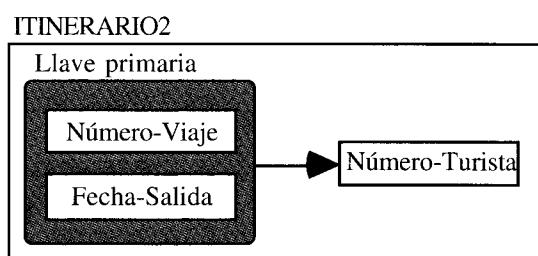
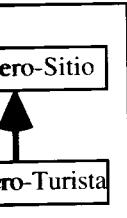


Fig. 5.14 continua.

e de la llave y por lo tanto no
13.



está en 3FN

era forma normal, se presentan

r de Número-Turista, Número-
ado de Número-Viaje, Fecha-

. Fecha-Salida se puede perder

urista, Número-Sitio, se puede
ndencia presente.
rema de descomposición a la
s relaciones

-Turista)

ero-Turista

Fig. 5.14 continua

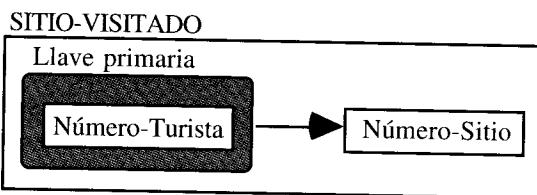


Figura 5.14. Ejemplo de relaciones en 3FN

5.6 TERCERA FORMA NORMAL BOYCE-CODD (3FNBC)

Una relación se encuentra en *tercera forma normal Boyce-Codd (3FNBC)* si todos los atributos son determinados solo por llaves, es decir, cada vez que

$$X \rightarrow A, A \notin X,$$

se verifica en R, entonces X contiene una llave de R.

Ejemplo. Considerar la relación ITINERARIO2 y suponer que se tiene la siguiente regla de integridad: "Cada turista tiene una sola fecha de salida". Esta regla de integridad se puede traducir por medio de la siguiente dependencia funcional:

$$\text{Número-Turista} \rightarrow \text{Fecha-Salida}$$

según se aprecia en la figura 5.15.

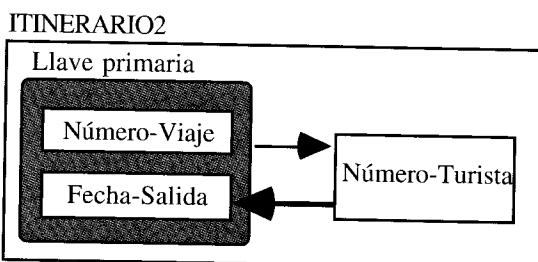


Figura 5.15 Ejemplo de relación que no está en 3FNBC

Así, ITINERARIO2 no se encuentra en 3FNBC. En esta situación aun persisten anomalías de actualización. En efecto, si por ejemplo, se desea suprimir un número de viaje en una fecha dada, se puede perder la información sobre cuándo el turista hará el viaje. Entonces, para resolver estos problemas, se aplica el teorema de descomposición a la dependencia funcional Número-Turista → Fecha-Salida, para obtener las dos relaciones siguientes:

ITINERARIO3(Número-Viaje, Número-Turista)
SALIDA-TURISTA(Número-Turista, Fecha-Salida),

según se aprecia en la figura 5.16.

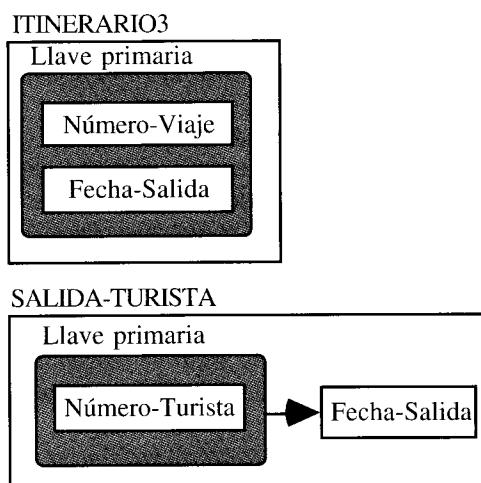


Figura 5.16 Ejemplo de relaciones en 3FNBC

5.7 DEPENDENCIAS MULTIVALUADAS Y LA CUARTA FORMA NORMAL (4FN)

Muchas personas dedicadas al análisis y diseño de bases de datos relacionales afirman que es suficiente contar con una base de datos cuyas relaciones se encuentren en 3FNBC pues se contaría con un alto grado de consistencia semántica. Sin embargo, existen ejemplos de relaciones

que, a pesar de estar en 3FNBC, tienen problemas de actualización.

Ejemplo:

DISPOSICIÓN

sobre la tabla de determinación. En la figura

Se puede conformar una relación para almacenar la información de la ciudad en la que se sabe, con un turista. Suponga que Pierre está en la ciudad de San Francisco. La inserción de la tupla (Pierre, San Francisco) en la relación Ciudad violaría la restricción de unicidad de la llave primaria Turista. Además, ¿qué sucede si se elimina la tupla (Pierre, Paris)?

situación aun persisten
se suprimir un número
n sobre cuándo el turista
se aplica el teorema de
sta → Fecha-Salida, para

que, a pesar de encontrarse en 3FNBC, pueden provocar problemas de almacenamiento, como se verá en el siguiente ejemplo.

Ejemplo. Considerar la relación siguiente:

DISPONIBILIDAD(Nombre-Turista, Continente, Ciudad)

sobre la disponibilidad de los turistas del Club de Ecoturismo en visitar sitios en determinados continentes y las ciudades de las cuales pueden salir.
En la figura 5.17, se aprecia un ejemplo de una tal relación.

DISPONIBILIDAD

NOMBRE-TURISTA	CONTINENTE	Ciudad
Carlos	América	Lima
Carlos	América	Caracas
Carlos	América	México
Pierre	América	México
Pierre	América	Bogotá
Carlos	Africa	Lima
Carlos	Africa	Caracas
Carlos	Africa	México
Pierre	Asia	México
Pierre	Asia	Bogotá

Figura 5.17 Ejemplo de relación que no está en 4FN

Se puede ver que esta relación se encuentra en 3FNBC, pues la llave primaria la conforman todos los atributos. Sin embargo, se pueden notar varias anomalías de almacenamiento.

En efecto, por ejemplo, no es posible insertar un continente y un turista sin conocer la ciudad de la cual puede salir, pues todos los atributos conforman la llave y como se sabe, no es permitido tener valores nulos en una llave primaria. Lo mismo ocurre con un turista y una ciudad, sin conocer el continente que se puede visitar.

Suponga que se va a insertar la tupla [Pierre, América, Santiago]. Pero, puesto que Pierre está también disponible para visitar Asia y como ahora puede salir también desde Santiago, entonces la inserción de la tupla anterior, debe desencadenar la inserción de la tupla [Pierre, Asia, Santiago].

Además, ¿qué ocurre si se deben suprimir los continentes visitados por un turista en particular ? En este caso, se suprime también el conocimiento que se tiene entre los

turistas y las ciudades. Finalmente, si una ciudad de un turista cambia, se debe hacer la modificación en varias tuplas.

El problema con esta relación es que no se encuentra en la cuarta forma normal.

Sin embargo, previo a la introducción del concepto de cuarta forma normal, se requiere la definición de lo que se entiende por dependencias multivaluadas. Estas se pueden ver como una generalización de las dependencias funcionales.

En efecto, se puede decir que una dependencia multivaluada existe entre dos conjuntos de atributos X y Y , y se denota por

$$X \rightarrow\rightarrow Y$$

si solo el conocimiento de X , e independientemente de otros atributos, determina un conjunto de valores relativos a Y .

Formalmente, sean X, Y subconjuntos de atributos de la relación R .

Se dice que la *dependencia multivaluada* $X \rightarrow\rightarrow Y$ se verifica si cada vez que las tuplas

$$[x_1, y_1, z_1] \text{ y } [x_2, y_2, z_2]$$

se encuentran en R , entonces deben estar también las tuplas

$$[x_1, y_2, z_1] \text{ y } [x_2, y_1, z_2]$$

En forma análoga a las dependencias funcionales, se puede introducir sobre las dependencias multivaluadas, una serie de axiomas de inferencia. Estas reglas fueron introducidas por C. Berri, R. Fagin y J.H. Howard en [BERR77] y se presentan a continuación.

Axiomas de inferencia de las dependencias multivaluadas

Sean X, Y, V, W subconjuntos de atributos de una relación R , y Z el complemento de los atributos de $X \cup Y$ con respecto a R . Entonces se verifican las siguientes reglas:

1. *Reflexividad*

2. *Complejidad*

3. *Aumento*

4. *Unión*

5. *Transitividad*

6. *Pseudotrasformación*

7. *Descomposición*

Por otra parte,

anunciar un teorema

multivaluadas:

Sea una relación R y sea X un subconjunto de sus atributos. La dependencia multivaluada $X \rightarrow\rightarrow Y$ se verifica si y solo si $R[X, Y] = R[X, Z, Y]$

Por su parte, se

forma normal (4)

dependencia multivaluada

una llave de R .

urista cambia, se debe hacer

encuentra en la cuarta

ncepto de cuarta forma
iende por dependencias
eneralización de las

cía multivaluada existe
ta por

ente de otros atributos,

utos de la relación R.

$\rightarrow Y$ se verifica si cada

las tuplas

les, se puede introducir
serie de axiomas de
C. Berri, R. Fagin y J.H.
ción.

ivaluadas

e una relación R, y Z el
pecto a R. Entonces se

1. *Reflexividad* $X \rightarrow\rightarrow X$
2. *Complemento* $X \rightarrow\rightarrow Y \Rightarrow X \rightarrow\rightarrow Z$
3. *Aumento* $X \rightarrow\rightarrow Y \text{ y } V \subseteq W \Rightarrow X \cup W \rightarrow\rightarrow Y \cup V$
4. *Unión* $\{X \rightarrow\rightarrow Y, X \rightarrow\rightarrow Z\} \Rightarrow X \rightarrow\rightarrow Y \cup Z$
5. *Transitividad* $\{X \rightarrow\rightarrow Y, Y \rightarrow\rightarrow Z\} \Rightarrow X \rightarrow\rightarrow Z - Y$
6. *Pseudotransitividad* $\{X \rightarrow\rightarrow Y, Y \cup W \rightarrow\rightarrow Z\} \Rightarrow X \cup W \rightarrow\rightarrow Z - (Y \cup W)$
7. *Descomposición* $\{X \rightarrow\rightarrow Y, X \rightarrow\rightarrow Z\} \Rightarrow \{X \rightarrow\rightarrow Y \cap Z, X \rightarrow\rightarrow Y - Z, X \rightarrow\rightarrow Z - Y\}$

Por otra parte, al igual que las dependencias funcionales, se puede enunciar un teorema de descomposición para el caso de las dependencias multivaluadas:

Teorema de Descomposición:

Sea una relación $R(X,Y,Z)$, en donde X, Y y Z son conjuntos de atributos y tal que la dependencia multivaluada $X \rightarrow\rightarrow Y$ se verifica en R. Entonces R se puede descomponer en las relaciones $R_1 = R[X, Y]$ y $R_2 = R[X, Z]$, es decir, $R = R_1 * R_2$

Por su parte, se dice que una relación $R(X,Y,Z)$ se encuentra en *cuarta forma normal (4FN)* si se encuentra en 3FNBC y si cada vez que una dependencia multivaluada $X \rightarrow\rightarrow Y$ se verifica, entonces X contiene a una llave de R.

Con respecto al ejemplo anterior, se puede ver que la dependencia multivaluada Nom-Turista →→ Ciudad-Salida se verifica en la relación DIPONIBILIDAD y por lo tanto no se encuentra en 4FN. Si se aplica el teorema de descomposición a esta dependencia multivaluada, se obtienen las dos relaciones que aparecen en la figura 5.18.

DISP-TURISTA		CIUDAD-TURISTA	
Nombre-Turista	Continente	Nombre-Turista	Ciudad
Carlos	América	Carlos	Lima
Pierre	América	Carlos	Caracas
Carlos	Africa	Carlos	México
Pierre	Asia	Pierre	México
		Pierre	Bogotá

Figura 5.18 Ejemplo de descomposición de relaciones en 4FN

5.8 QUINTA FORMA NORMAL (5FN)

La quinta forma normal (5FN) se refiere a las llamadas dependencias producto que garantizan la descomposición de una relación en tres o más relaciones, manteniendo el contenido original y con menor redundancia.

Formalmente, sean X_1, X_2, \dots, X_n , subconjuntos de atributos de una relación R, en donde la unión es igual a los atributos de R. Se dice que la *dependencia producto de orden n*, denotada por

$$*[X_1][X_2]\dots[X_n]$$

se verifica en R si

$$R = R[X_1] * R[X_2] * \dots * R[X_n]$$

Una relación R se dice que se encuentra en *quinta forma normal (5FN)* si cada dependencia producto $*[X_1][X_2]\dots[X_n]$ de R está inducida

por las llaves candidata •

Ejemplo...

GUIA(Nc)

que traduc... presenta e

En este cas...
una serie de...
En efecto, s...
debe insertar...
Irazú, Miam...
Con respect...
Juan, Irazú...
contrario, si...
problema es

* [Nom-Guf...
y esta depen...
Esto es, la re...
si así se hici...
la figura 5.20

Por lo tanto,...
relaciones, seg...

r que la dependencia
a se verifica en la
encuentra en 4FN. Si
a esta dependencia
que aparecen en la

TURISTA

Ciudad
Lima
Caracas
México
México
Bogotá

ones en 4FN

llamadas dependencias
relación en tres o más
on menor redundancia.
os de atributos de una
utos de R. Se dice que

quinta forma normal
X_n] de R está inducida

por las llaves candidatas de R, es decir, cada X_i contiene una llave candidata de R.

Ejemplo. Considerar la siguiente relación:

GUIA(Nombre-Guía, Nombre-Sitio, Ciudad-Salida)

que traduce los desplazamientos de los Guías del Club de Ecoturismo y que se presenta en la figura 5.19.

GUIA

Nombre-Guía	Nombre-Sitio	Ciudad-Salida
Juan	Irazú	San José
Juan	Tikal	Miami
Ana	Irazú	Miami
Juan	Irazú	Miami

Figura 5.19 Ejemplo de una relación que no está en 5FN

En este caso, se puede ver que la relación se encuentra en 4FN. Aún así, persisten una serie de anomalías.

En efecto, si por ejemplo se desea introducir la tupla [Ana, Irazú, Miami], también debe insertarse la tupla [Juan, Irazú, Miami]. Pero, si se introduce la tupla [Juan, Irazú, Miami] esto no condiciona nada.

Con respecto a las supresiones, el problema es contrario. Así, si se borra la tupla [Juan, Irazú, Miami], entonces se debe suprimir [Ana, Irazú, Miami]. Por el contrario, si se suprime la tupla [Juan, Irazú, Miami] esto no condiciona nada. El problema es que se verifica la dependencia producto:

- * [Nom-Guía,Nom-Sitio], [Nom-Sitio,Ciudad-Sal], [Nom-Guía,Ciudad-Sal].

y esta dependencia producto no está inducida por las llaves de la relación GUIA. Esto es, la relación GUIA no se puede descomponer en solo dos relaciones, ya que si así se hiciera se generarían tuplas ajenas a la relación original, como lo muestra la figura 5.20.

Por lo tanto, en este caso es recomendable descomponer la relación GUIA en las tres relaciones, según la descomposición producto presente, como lo muestra la figura 5.21.

GUIA[N-G, N-S] * GUIA[N-S,C-S]

Nom-Guía	Nom-Sitio	Ciudad-Salida
Juan	Irazú	San José
Juan	Irazú	Miami
Juan	Tikal	Miami
Ana	<i>Irazú</i>	<i>San José</i>
Ana	Irazú	Miami

GUIA[N-S, C-S] * GUIA[N-G,C-S]

Nom-Guía	Nom-Sitio	Ciudad-Salida
Juan	Irazú	San José
Juan	Irazú	Miami
Juan	Tikal	Miami
Ana	<i>Tikal</i>	<i>San José</i>
Ana	Irazú	Miami

Figura 5.20 Ejemplo de tuplas ajenas

GUIA[N-G, N-S]

Nom-Guía	Nom-Sitio
Juan	Irazú
Juan	Tikal
Ana	Irazú

GUIA[N-S,C-S]

Nom-Sitio	Ciudad-Salida
Irazú	San José
Tikal	Miami
Irazú	Miami

GUIA[N-G, C-S]

Nom-Guía	Ciudad-Salida
Juan	San José
Juan	Miami
Ana	Miami

Figura 5.21 Ejemplo de relaciones en 5FN

EJERCICIOS Y PREGUNTAS DE REPASO

5.1 Sea el siguiente conjunto de atributos, que traducen un sistema de utilización de un laboratorio de computadores.

- #U (Número de usuario)
- NU (Nombre de usuario)

DU (D)
HI (E)
HF (F)
FU (G)
#E (M)
NE (N)
CE (C)
NP (L)
DP (D)
TM (T)
FV (F)
y las :

Utiliza
de esqu

5.2 Dar un
encuen

5.3 Sea el
#PLAC
chofer"

¿Se enc

DU (Dirección de usuario)
 HI (Hora inicial de uso del equipo)
 HF (Hora final de uso del equipo)
 FU (Fecha de uso del equipo)
 #E (Número de equipo)
 NE (Nombre del equipo)
 CE (Capacidad del equipo)
 NP (Nombre del proveedor)
 DP (Dirección del proveedor)
 TM (Tipo de mantenimiento del proveedor)
 FV (Fecha de venta del equipo)

y las siguientes dependencias funcionales

$$\begin{aligned} \#U &\rightarrow NU \\ \#U &\rightarrow DU \\ \#E &\rightarrow NE \\ \#E &\rightarrow CE \\ NP &\rightarrow DP \\ NP &\rightarrow TM \\ FV, NP, \#E &\rightarrow TM \\ \#U, HI, HF, FU &\rightarrow \#E \end{aligned}$$

Utilizando el teorema de descomposición, encontrar un conjunto de esquemas que se encuentren en 3FNBC.

- 5.2 Dar un ejemplo de un esquema que esté en 3FN pero que no se encuentre en 3FNBC.
- 5.3 Sea el esquema CHOFER(#Emp, Nom-Emp, Dirección, Salario, #PLACA) y la regla “cada camión será conducido solo por un chofer”.

¿Se encuentra este esquema en 3FNBC? Justifique su respuesta.

en 5FN

que traducen un sistema de
adadores.

Salida
é
nas

S]

Salida
é
se

enas

GUIA[N-G, C-S]

Nom- Guía	Ciudad- Salida
Juan	San José
Juan	Miami
Ana	Miami

- 5.4 Sea el esquema $R(\#Usuario, Fecha, Hora-Inicial, \#Equipo)$ y el conjunto de dependencias funcionales
 $\{ (\#Usuario \rightarrow Fecha), (\#Usuario, Fecha, Hora-Inicial \rightarrow \#Equipo) \}$.
 ¿Por qué este esquema es incorrecto?
- 5.5 Sea $R(A, B, C, D, E, F)$ y el siguiente conjunto de dependencias funcionales $F = \{ (A, B \rightarrow C), (A, D \rightarrow C, E), (A, E \rightarrow F) \text{ y } (E, D \rightarrow F) \}$. Determinar una llave primaria para este esquema.
- 5.6 Mostrar las reglas A4, A5 y A6, de la sección 5.3.2
- 5.7 Dado el esquema $R(A, B, C, D)$ y el siguiente conjunto de dependencias funcionales $F = \{ A \rightarrow B \text{ y } B \rightarrow C \}$, determinar cuáles de las siguientes dependencias funcionales se pueden derivar de F, usando las reglas de Armstrong.
- $A \rightarrow C$
 - $A \rightarrow B; C$
 - $A, B, C \rightarrow D$
 - $A, D \rightarrow B, C$.
- 5.8 Sea el esquema $R(A, B, C, D, E, F, G, I)$ y el conjunto de dependencias funcionales
 $F = \{ (A \rightarrow C), (B, H \rightarrow E, G), (A, H, I \rightarrow F) \}$
 Usando el algoritmo de la cerradura, determinar $\{A, H, I\}^+$.
- 5.9 Mostrar que para dos conjuntos de dependencias funcionales F y G, se tiene que
- $F^+ = (F^+)^+$.
 - Si $F \subseteq G$ entonces $F^+ \subseteq G^+$.
- 5.10 Determinar una cobertura mínima para el siguiente conjunto de dependencias funcionales
 $F = \{ (A, B \rightarrow C, G), (B, H \rightarrow E, G), (B \rightarrow C), (H \rightarrow B, C) \}$.

5.11 Mostrar que si se cumple la condición multivaluada en un esquema

5.12 Sean los esquemas $R_1 = VIAJE$ y $R_2 = VIAJE$

a. $A, B \rightarrow C$

b. $A \rightarrow C$

c. $A \rightarrow C$

5.13 Sea el esquema $R_3 = VIAJE$

VIAJES

y la siguiente relación

a. ¿Se encue

Sean las sig

R1 = VIAJE

R2 = VIAJE

R3 = VIAJE

b. Efectuar $R_1 * R_2$

$R_1 * R_2$

$R_1 * R_3$

$R_1 * R_2 * R_3$

c. ¿Se encue

-Inicial, #Equipo) y el
o}.

junto de dependencias
(E). (A,E → F) y (E,D →
este esquema.

ión 5.3.2

siguiente conjunto de
y B → C}, determinar
funcionales se pueden
ng.

conjunto de dependencias

F)

minar {A,H,I}+.

ndencias funcionales F y

el siguiente conjunto de

→ C), (H → B,C) }.

5.11 Mostrar las reglas de inferencia aplicadas a las dependencias multivaluadas.

5.12 Sean los atributos A, B, C y D. Dar ejemplos de relaciones con al menos cuatro tuplas, en donde se verifique:

- a. A,B →→ C y C → D
- b. A → B y B →→ C
- c. A → B y B,C → D

5.13 Sea el esquema

VIAJE(Número-Turista, Número-Sitio, C-Salida, C-Llegada)
y la siguiente relación

VIAJE

N-Turista	N-Sitio	C-Salida	C-Llegada
200	400	Kampala	Nairobi
100	400	Nairobi	Kampala
100	400	Kampala	Nairobi
100	500	Kampala	Nairobi

a. ¿Se encuentra esta relación en 4FN? Justifique su respuesta.

Sean las siguientes proyecciones

R1 = VIAJE[Número-Turista, Número-Sitio]

R2 = VIAJE[Número-Turista, C-Salida, C-Llegada]

R3 = VIAJE[Número-Sitio, C-Salida, C-Llegada]

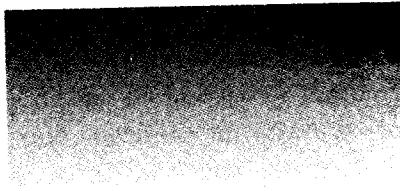
b. Efectuar los siguientes productos

R1 * R2

R1 * R3

R1 * R2 * R3

c. ¿Se encuentra en 5 FN? Justifique su respuesta.



6

Modelaje y Diseño de Bases de Datos

«Desde distintos puntos de vista dos hombres miran el mismo paisaje. Sin embargo, no ven lo mismo. La distinta situación hace que el paisaje se organice ante ambos de distinta manera. Lo que para uno ocupa el primer término y acusa con vigor todos sus detalles, para el otro se halla en el último y queda oscuro y borroso. Además, como las cosas puestas unas detrás de otras se ocultan en todo o en parte, cada uno de ellos percibirá porciones del paisaje que al otro no llegan. ¿Tendría sentido que cada cual declarase falso el paisaje ajeno? Evidentemente, no; tan real es el uno como el otro. Pero tampoco tendría sentido que puestos de acuerdo, en vista de no coincidir sus paisajes, los juzgasen ilusorios. Esto supondría que hay un tercer paisaje auténtico, el cual no se halla sometido a las mismas condiciones que los otros dos.»

José Ortega y Gasset (1883-1955),
escritor español.

6.1 INTRODUCCION

Una de la tareas más importantes en el desarrollo de un sistema de información es el referente al modelaje y diseño de la base de datos. El *modelaje* se refiere a una técnica que permite interpretar los requerimientos de información de una organización. Como resultado de aplicar esta técnica se obtiene un esquema conceptual. Este esquema por su parte, se transforma en un conjunto de estructuras relacionales y mecanismos que aseguran la integridad de la base de datos. Esta transformación se conoce como *diseño de la base de datos*.

En el presente capítulo se introduce una herramienta -metodología- que permite el modelaje y el diseño de bases de datos. Previamente, se hace un rápido recorrido sobre varios tipos de metodologías de desarrollo de sistemas de información que se han generado en los últimos 25 años.

6.2 CLASIFICACION DE METODOLOGIAS

Hasta la fecha se puede hablar de tres grandes generaciones de metodologías de desarrollo de sistemas:

- Metodologías orientadas a los procesos
- Metodologías orientadas a los datos
- Metodologías orientadas a los objetos

6.2.1 Metodologías orientadas a los procesos

La aparición de las nuevas metodologías de desarrollo de sistemas se inicia a finales de la década de 1960-1970 con las llamadas *metodologías orientadas a los procesos*. La necesidad de contar con estas herramientas se explica debido a que los sistemas cada día crecían en complejidad. El concepto de *ciclo de vida* de un sistema de información data de esta época. Con tal herramienta, el mundo real se representa por medio de diagramas de flujo de datos -cuyo objetivo es mostrar en forma gráfica

el movimiento de los datos a través de un sistema-, especificaciones de los procesos y un diccionario de datos.

La primera metodología de este tipo se debe a Tom DeMarco [DEMA78], quien la denominó *análisis estructurado*. Esta es una herramienta acorde con los principios de la ingeniería del software de la modularidad, módulos débilmente acoplados y cohesión de módulos. A pesar de que el modelaje de datos no fue desconocido en esta generación, el énfasis se centró en el proceso de modelaje.

Las personas que utilizaban el análisis estructurado de DeMarco se enfrentan con muchos problemas a la hora de escoger las burbujas y los respectivos diagramas de flujo. Debido a la evolución de los sistemas, así como de los lenguajes de programación, E. Yourdon actualiza el análisis estructurado con el nombre de *análisis estructurado moderno*.

La diferencia fundamental de estas dos propuestas radica en que el análisis estructurado moderno adiciona una fase previa con el fin de desarrollar un “modelo esencial” del sistema y enfatiza mucho más en el modelaje de los datos -por medio de diagramas de entidad-asociación- y el modelaje del comportamiento -por medio de diagramas de estado-transición- [FICH92].

Estos hechos sirven de puente para establecer otro grupo de metodologías las cuales se denominan orientadas a los datos.

6.2.2 Metodologías orientadas a los datos

Un cambio importante se da con la aparición de las *metodologías orientadas a los datos*, las cuales se desarrollaron sobre la base del modelo de datos relacional y el modelo de datos entidad-asociación. La herramienta principal de tales metodologías es el *diagrama de entidad-asociación*, en donde se expresan las posibles conexiones -asociaciones- entre conjuntos de datos -entidades-. En este enfoque, el proceso de diseño gira alrededor de los datos y sus propiedades.

El enfoque orientado a los datos parte de los siguientes supuestos [FICH92]:

- Los datos o el diseño de
- Los datos independie

En la actualid
alizadas por su s
parte del usuario y

6.2.3 Metodologí

A mediados
metodologías llam
se basan en el par

Contrario a las
nen la particula
así como su comp
de estándares en
cantidad de propu
en entre las me
mucho depende de
metodologías a

Algunas de est
l dedicado a la
paradigma de la o

En el cuadro
desarrollo de sist

A pesar de la g
ra el desarrollo
presentación fie
conjuntos de regla
garantizarse la
ño. El grado de

a-, especificaciones de

lebe a Tom DeMarco
ucturado. Esta es una
iería del software de la
ohesión de módulos. A
cido en esta generación.

cturado de DeMarco se
coger las burbujas y los
lución de los sistemas.
Yourdon actualiza el
structurado moderno.

uestas radica en que el
e previa con el fin de
afatiza mucho más en el
de entidad-asociación- y
diagramas de estados-

blecer otro grupo de
s a los datos.

n de las *metodologías*
aron sobre la base del
entidad-asociación. La
l diagrama de entidad-
nexiones -asociaciones-
enfoque, el proceso de
ndes.

os siguientes supuestos

- Los datos organizacionales brindan un fundamento más sólido para el diseño de un sistema que los procedimientos organizacionales.
- Los datos pueden verse como un recurso organizacional independiente de los sistemas que procesan los datos.

En la actualidad las metodologías orientadas a los datos son muy utilizadas por su simplicidad de representación, su fácil comprensión por parte del usuario y por permitir un traslado muy natural al mundo relacional.

6.2.3 Metodologías orientadas a los objetos

A mediados de la década de 1980-1990 surgen las primeras metodologías llamadas *metodologías orientadas a los objetos*, las cuales se basan en el paradigma de orientación a objetos.

Contrario a las metodologías tradicionales, las orientadas a objetos tienen la particularidad de representar en una misma estructura los datos así como su comportamiento. Sin embargo, no existe consenso por falta de estándares en el paradigma de orientación a objetos y existe gran cantidad de propuestas disímiles entre ellas. Así, las diferencias que se dan entre las metodologías de este tipo, pueden ser muy grandes y mucho depende del interés de los autores en rescatar parte de la filosofía de metodologías anteriores e incorporarlas a sus nuevas propuestas.

Algunas de estas metodologías se estarán estudiando en el capítulo 14, dedicado a la tecnología de bases de datos sustentada sobre el paradigma de la orientación a objetos.

En el cuadro 6.1 se presentan algunas de las metodologías de desarrollo de sistemas más importantes.

A pesar de la gran cantidad de metodologías que se pueden utilizar para el desarrollo de un sistema de información, ninguna permite una representación fiel del mundo real. Estas metodologías siguen siendo conjuntos de reglas generales que sirven de guía a los analistas con el fin de garantizarse las características mínimas de un buen modelaje y diseño. El grado de fidelidad con que se pueda representar el mundo real,

no solo depende de la metodología utilizada, sino también de la experiencia y la capacidad del analista.

Sin embargo, es un imperativo que las organizaciones tengan como política la utilización de alguna metodología. Muy a menudo, por no utilizar dichas herramientas los sistemas desarrollados producen gran cantidad de errores y, en consecuencia, resultan poco confiables.

En este capítulo se introduce una metodología general para el modelaje y diseño de bases de datos, basada en un modelo orientado a los datos.

Esta metodología persigue representar en forma conceptual lo que se denomina un *esquema de aplicación* que se define como una representación en forma de datos de los requerimientos que se establecen para modelar una función particular -administrativa, CAD/CAM, etc.-. Ejemplos de estas funciones son: el pago de cheques de una empresa, el sistema eléctrico de un edificio, el sistema de alquiler de una agencia dedicada al alquiler de automóviles, etc.

Cuadro 6.1 Metodologías de desarrollo de sistemas

Año	Nombre	Orientad a	Autores
1974	SA/SD	Procesos	DeMarco, Yourdon
1975	JSD	Datos	Jackson
1977	SADT	Procesos	Roos
1979	Merise	Datos	Tardieu
1982	NIAM	Datos	Nijssen
1983	OOD	Objetos	Booch
1985	SA-RT/SD-RT	Procesos	Ward, Mellor
1987	HATLEY	Procesos	Hatley, Pirbhai
1989	OOA/OOD	Objetos	Coad, Yourdon
1989	OOSD	Objeto	Wasserman et al.
1989	OORS	Objetos	Bailin
1990	IE	Datos	Martin
1991	OMT	Objetos	Rumbaugh
1991	OSA	Objetos	Embley et al.

Establece complejas de datos para empresas limitadas.

Por esas cuales conceptos de aplicación integración.

A continuación se parte de lo presentado.

6.3 DICCIÓNARIOS

Una herencia de la creación.

Un diccionario que brinda almacenamiento de datos es actualizable permanentemente.

Así, el control de los cambios.

Los dos tipos:

- Permiten a los usuarios definir sus propios términos.
- Ayudan a organizar solo lugares.

sino también de la
aciones tengan como
y a menudo, por no
lados producen gran
co confiables.

general para el modelaje
entado a los datos.

conceptual lo que se
define como una
tos que se establecen
ra, CAD/CAM, etc.-.
es de una empresa, el
uiler de una agencia

Establecer esquemas de aplicación es una necesidad por razones de complejidad. En efecto, si lo que se desea es modelar y diseñar una base de datos que traduzca las diferentes funciones administrativas de una empresa, este proceso se puede volver inmanejable, debido a limitaciones propias de los seres humanos.

Por esta razón, es buena política trabajar con funciones particulares, las cuales, una vez diseñadas, se integrarán en un solo esquema conceptual global. Este proceso se denomina *integración de esquemas de aplicación*. En el próximo capítulo se introduce una metodología de integración de esquemas de aplicación.

A continuación se introduce el concepto de diccionario de datos, que es parte importante en la metodología de modelaje y diseño que aquí se presenta.

3 DICCIONARIO DE DATOS

Una herramienta fundamental para el buen diseño de bases de datos es la creación y el mantenimiento en paralelo de un diccionario de datos.

Un *diccionario de datos* es un repositorio, manual o automatizado, que brinda la información sobre los datos que se encuentran almacenados en una base de datos. Por lo general, un diccionario de datos es a su vez una base de datos y por lo tanto se puede almacenar y manipular por medio de un sistema computacional.

Así, el diccionario de datos tiene una doble misión de organización y control de los datos. Este es el núcleo de cualquier esquema conceptual.

Los dos grandes objetivos de un diccionario de datos son:

- Permitir la administración de la documentación de los datos, y a los usuarios entender lo que son los datos y qué significan las definiciones y descripciones.
- Ayudar a controlar mejor los recursos de datos, ya que con un diccionario de datos la información sobre ellos permanece en un solo lugar.

Un diccionario de datos debe al menos contar con la siguiente información:

- Nombres de las relaciones y los atributos que la conforman.
- Las llaves primarias, alternas, externas de las relaciones
- Para cada atributo se debe establecer una serie de conceptos, como por ejemplo:
 - tipo de dato
 - formato
 - rango
 - significado
 - valores permitidos
 - unicidad
 - soporte de nulos
- Las reglas de integridad y operaciones de disparo, etc.

Un diccionario de datos puede ser dependiente de un SABD o independiente del mismo.

Un diccionario de datos *independiente* es aquel creado por ejemplo, por compañías consultoras y emplean un SABD para administrar sus *meta-datos* o brindan sus propios medios para almacenar y recuperar información. Un diccionario de datos de este tipo con frecuencia sirve de interfaz a varios SABDs.

Por otra parte, un diccionario de datos *dependiente* forma parte del SABD y lo emplea para administrar sus datos.

En cuanto a los beneficios de usar un diccionario de datos, se pueden mencionar los siguientes:

- Lograr homogeneidad en la representación de los datos.
- Lograr concordancia en las definiciones de los datos.
- Generar automáticamente la descripción de los datos.
- Reducir el costo del desarrollo del sistema.

s contar con la siguiente
os que la conforman.
de las relaciones
na serie de conceptos, como

de disparo, etc.
pendiente de un SABD o
s aquel creado por ejemplo,
SABD para administrar sus
ara almacenar y recuperar
tipo con frecuencia sirve de

dependiente forma parte del
s.
cionario de datos, se pueden
ión de los datos.
s de los datos.
n de los datos.
ma.

- Cumplir con los estándares de los datos.
- Eliminar la redundancia de los datos.
- Mejorar el control en los cambios.
- Reducir el esfuerzo de análisis de los datos.
- Ayudar en la comunicación.
- Ayudar en el desarrollo paralelo.
- Bajar los costos de administración.
- Mejorar la seguridad.

Una metodología de diseño debe acompañarse de una documentación confiable y estándar en un diccionario de datos. Este es un imperativo que debe tomarse en consideración a la hora que se deseé modelar y diseñar una base de datos, independiente de la complejidad de la misma.

Resumiendo, se puede decir que el uso de un diccionario de datos es un imperativo para el buen funcionamiento en un ambiente de bases de datos. Este puede beneficiar considerablemente a una organización, pero solo en el caso en que se haya seguido un procedimiento correcto.

6.4 METODOLOGIA DE DISEÑO

En esta sección se presenta la metodología de diseño de bases de datos. La misma se sustenta en el modelo de datos introducido en el capítulo 2. Esta metodología está constituida de una serie de etapas, las cuales sirven como una guía genérica para que el diseñador pueda concebir sus bases de datos

Una de las características de esta metodología es que permite una comunicación fluida con el usuario. Esto es determinante para el éxito del proceso de diseño, ya que de esta comunicación puede depender que al usuario se le brinde posteriormente lo que él realmente requiere.

Para explicar dicha metodología, se introduce un ejemplo, el cual se irá desarrollando en las diferentes etapas de la misma.

Ejemplo. La compañía CG Systems se dedica a la venta de computadores personales. Se desea modelar y diseñar una base de datos que permita llevar el control de las ventas anuales de los tipos de microcomputadores que son distribuidos por esta compañía.

6.4.1 Establecer las entidades

El primer paso en la metodología es el establecimiento de las entidades que son de interés a la aplicación que se desea modelar.

Si el análisis y diseño de un sistema de información es una actividad complicada, mucho de esto se debe a lo difícil que significa determinar las entidades pertinentes a la aplicación. En efecto, debido a la gran cantidad de variables que entran en juego para representar el mundo real: incertidumbre, tiempo, espacio, caos, etc., lo único que se puede hacer es representar un instante -una fotografía- de dicho mundo real, y esto, independiente del tipo de metodología utilizada. Sin embargo, lo que sí se puede hacer es utilizar una metodología que pueda representar en forma más natural dicho mundo real o dominio del problema. Varios autores [FLEM89], [COAD91], [BATI92], [EMBL92], etc., proponen una serie de reglas generales que ayudan a determinar los elementos implicados en la aplicación, llámense éstos relaciones, entidades o clases de objetos.

Ejemplo (continuación).

Al estudiar el dominio del problema se pueden establecer, junto con los usuarios, las entidades de objetos más relevantes para la aplicación. Después de discusiones con los usuarios, se establecen tres entidades principales:

CLIENTE
COMPUTADOR -tipo-
VENDEDOR

Por otra parte, al refinar aun más esta etapa se establece la importancia de diferenciar los clientes. De un lado, se tienen los clientes que adquieren un equipo en forma individual y cuyo pago debe ser de contado. Por otro lado, se tienen los clientes que

en ejemplo, el cual se
na. .

venta de computadores
os que permita llevar el
computadores que son

iento de las
odelar.

a actividad
minar las
cantidad
do real:

de hacer es
ndo real, y esto,
mbargo, lo que sí se
representar en forma
ema. Varios autores
proponen una serie
entos implicados en
lases de objetos.

junto con los usuarios,
Después de discusiones

importancia de diferenciar
en un equipo en forma
e tienen los clientes que

son empresas, las cuales adquieren microcomputadores al por mayor, lo que permite descuentos y créditos por parte de la compañía CG Systems. Así, esta diferenciación se hace necesaria por razones de precio, créditos, etc., y quizás sea conveniente introducir dos nuevas entidades:

CLIENTE-PERSONA
CLIENTE-EMPRESA

Además, la compañía CG Systems, por razones de mantenimiento y de garantía de los equipos que se venden, considera necesario diferenciar sus componentes fundamentales, a saber: teclado, CPU y monitor.

Por lo tanto, es recomendable establecer tres nuevas entidades que serán llamadas:

TECLADO
CPU
MONITOR

En la figura 6.1 se presentan las diferentes entidades que podrían conformar este esquema.

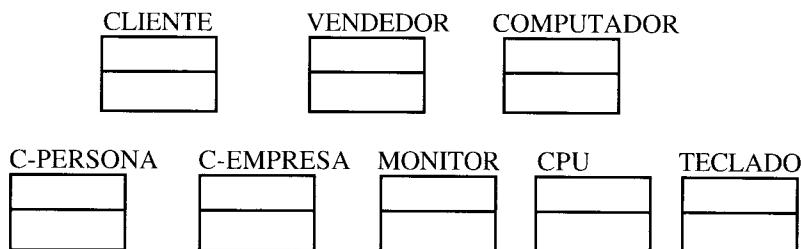


Figura 6.1 Entidades del esquema sobre la venta de microcomputadores

Si otro diseñador realiza esta etapa, es posible que obtenga otras entidades. Esto significa que no existe una única solución.

Al final de esta etapa, se tiene una representación gráfica de dichas entidades. Sin embargo, se debe recordar que además de esta representación, hay que crear un diccionario de datos que permita documentar lo que significan estas cajas.

Con el resto de las etapas se debe proceder de igual manera, es decir, toda representación gráfica debe tener su correspondiente representación en un diccionario de datos.

Para explicar dicha metodología, se introduce un ejemplo, el cual se irá desarrollando en las diferentes etapas de la misma.

Ejemplo. La compañía CG Systems se dedica a la venta de computadoras personales. Se desea modelar y diseñar una base de datos que permita llevar el control de las ventas anuales de los tipos de microcomputadores que se distribuidos por esta compañía.

6.4.1 Establecer las entidades

El primer paso en la metodología es el establecimiento de las entidades que son de interés a la aplicación que se desea modelar.

Si el análisis y diseño de un sistema de información es una actividad complicada, mucho de esto se debe a lo difícil que significa determinar las entidades pertinentes a la aplicación. En efecto, debido a la gran cantidad de variables que entran en juego para representar el mundo real -incertidumbre, tiempo, espacio, caos, etc., lo único que se puede hacer es representar un instante -una fotografía- de dicho mundo real, y esto independiente del tipo de metodología utilizada. Sin embargo, lo que sí se puede hacer es utilizar una metodología que pueda representar en forma más natural dicho mundo real o dominio del problema. Varios autores [FLEM89], [COAD91], [BATI92], [EMBL92], etc., proponen una serie de reglas generales que ayudan a determinar los elementos implicados en la aplicación, llámense éstos relaciones, entidades o clases de objetos.

Ejemplo (continuación).

Al estudiar el dominio del problema se pueden establecer, junto con los usuarios, las entidades de objetos más relevantes para la aplicación. Después de discusiones con los usuarios, se establecen tres entidades principales:

CLIENTE
COMPUTADOR -tipo-
VENDEDOR

Por otra parte, al refinar aun más esta etapa se establece la importancia de diferenciar los clientes. De un lado, se tienen los clientes que adquieren un equipo en forma individual y cuyo pago debe ser de contado. Por otro lado, se tienen los clientes que

empleo, el cual se

ta de computadores
que permita llevar el
computadores que son

ecimiento de las
ea modelar.

es una actividad
fica determinar las
a la gran cantidad
el mundo real:
se puede hacer es
ndo real, y esto,
bargo, lo que sí se
resentar en forma
na. Varios autores
roponen una serie
tos implicados en
ses de objetos.

nto con los usuarios.
spués de discusiones

ortancia de diferenciar
n un equipo en forma
ien los clientes que

son empresas, las cuales adquieren microcomputadores al por mayor, lo que permite descuentos y créditos por parte de la compañía CG Systems. Así, esta diferenciación se hace necesaria por razones de precio, créditos, etc., y quizás sea conveniente introducir dos nuevas entidades:

CLIENTE-PERSONA
CLIENTE-EMPRESA

Además, la compañía CG Systems, por razones de mantenimiento y de garantía de los equipos que se venden, considera necesario diferenciar sus componentes fundamentales, a saber: teclado, CPU y monitor.

Por lo tanto, es recomendable establecer tres nuevas entidades que serán llamadas:

TECLADO
CPU
MONITOR

En la figura 6.1 se presentan las diferentes entidades que podrían conformar este esquema.

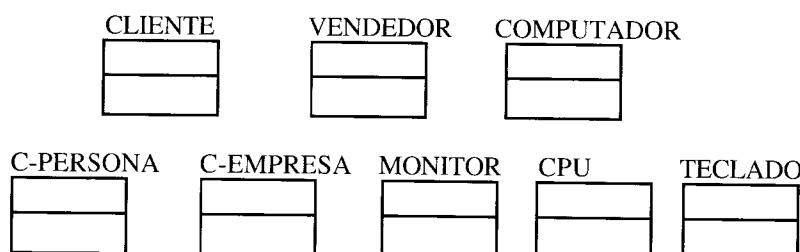


Figura 6.1 Entidades del esquema sobre la venta de microcomputadores

Si otro diseñador realiza esta etapa, es posible que obtenga otras entidades. Esto significa que no existe una única solución.

Al final de esta etapa, se tiene una representación gráfica de dichas entidades. Sin embargo, se debe recordar que además de esta representación, hay que crear un diccionario de datos que permita documentar lo que significan estas cajas.

Con el resto de las etapas se debe proceder de igual manera, es decir, toda representación gráfica debe tener su correspondiente representación en un diccionario de datos.

6.4.2 Establecer herencias y composiciones

En segundo lugar, se deben determinar las herencias y composiciones del esquema de aplicación. Con respecto a una herencia, se puede seguir el siguiente proceso:

1. Considerar cada entidad como una super-entidad potencial. Luego, valorar, cuáles de las entidades restantes pueden ser sub-entidades.
2. Si la entidad no califica como super-entidad, considerarla como una sub-entidad potencial y verificar si existe una posible super-entidad asociada.
3. Si la entidad no satisface ninguno de los dos puntos anteriores, entonces la entidad no es parte de ninguna herencia.

Por su parte, con respecto a una composición, se sigue un trámite similar, es decir,

1. Considerar la entidad como un compuesto potencial. Luego, valorar, cuáles de las entidades restantes pueden ser componentes asociados.
2. Si la entidad no califica como compuesto, considerarla como un componente potencial y verificar si existe un posible compuesto asociado.
3. Si la entidad no satisface ninguno de los dos puntos anteriores, entonces la entidad no es parte de ninguna composición.

Una vez que un conjunto de entidades califica dentro de una composición, se determinan las cardinalidades respectivas.

Se debe recordar que estas verificaciones son factibles, pues lo que se está modelando es una aplicación particular, en donde el número de entidades no supera varias decenas. En un sistema de varios cientos de entidades, esta situación se hace inmanejable. De ahí la importancia de reducir el problema del diseño de sistemas a sub-problemas que puedan ser manejados por el diseñador.

6.4.3 D

Una
esquem
siguen
existen

El e
caso, er

Ejemplo (continuación).

En este caso se puede establecer una herencia entre la entidad CLIENTE -super-entidad-, y las entidades CLIENTE-PERSONA y CLIENTE-EMPRESA -sub-entidades-.

De igual forma se puede proceder en el caso de las composiciones, es decir, con respecto a los compuestos y componentes. Es el caso que se da entre la entidad COMPUTADOR -compuesto- y las entidades TECLADO, CPU y MONITOR -componentes-.

Con respecto a las cardinalidades se puede ver que un compuesto -monitor, cpu o teclado- puede que no forme parte de un computador y a lo sumo puede pertenecer a un computador. Así, las cardinalidades con respecto a las entidades MONITOR, CPU y TECLADO son (0,1).

Por su parte, las cardinalidades con respecto al compuesto son (1,1), es decir, al menos y a lo sumo un computador se compone de un teclado, un cpu y un monitor. En la figura 6.2 se aprecia el esquema de aplicación después de haber cumplido con esta segunda etapa.

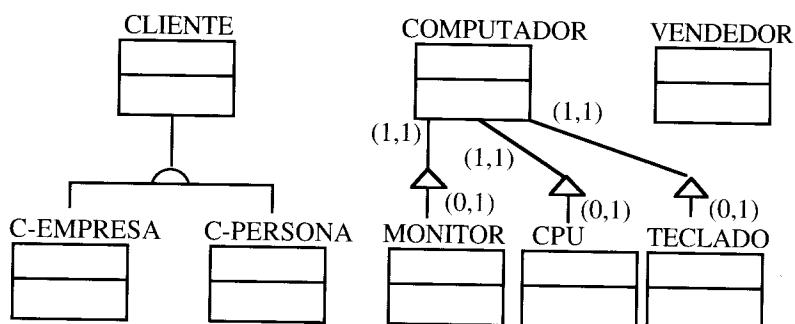


Figura 6.2 Herencias y composiciones del esquema sobre la venta de microcomputadores

6.4.3 Determinar las asociaciones

Una vez que se han determinado las entidades principales del esquema de aplicación, así como las herencias y composiciones, el siguiente paso en la metodología es determinar las asociaciones que existen entre estas entidades.

El establecer las asociaciones es una tarea también difícil. En este caso, en la fase de modelaje, cuando se discute con el usuario sobre el

alcance del sistema, un verbo o una preposición pueden darle la pista al usuario de que existe una asociación entre dos entidades.

Para cada asociación que se determina, se debe establecer:

- su tipo, es decir, 1-1, 1-N, N-1, M-N.
- su cardinalidad mínima y máxima.

La información de las asociaciones así como sus tipos y cardinalidades debe documentarse en el diccionario de datos.

Ejemplo (continuación).

Según los alcances de la aplicación, se determina que los clientes están interesados en *comprar* computadores. De aquí surge la idea de crear una asociación (*compra*) entre las entidades CLIENTE y COMPUTADOR y ésta es del tipo M-N, ya que un cliente puede interesarse en varios tipos de computadores y un tipo de computador puede interesar a varios clientes. Además deben calcularse las cardinalidades de esta asociación. En primer lugar, para considerar un cliente éste debe comprar al menos un tipo de computador o puede comprar varios. De ahí que la cardinalidad con respecto a la entidad CLIENTE es (1,n). En segundo lugar, un tipo de computador puede que no sea adquirido por nadie o por el contrario por varios clientes, entonces la cardinalidad con respecto a la entidad COMPUTADOR es (0,n).

Asimismo, existe una asociación entre las entidades COMPUTADOR y VENDEDOR que suplen los tipos de computador. A esta asociación se le llamará (*vende*) y también es del tipo M-N, ya que un tipo de computador puede ser vendido por varios vendedores y un vendedor puede vender varios tipos de computadores. Por su parte, la cardinalidad con respecto a la entidad COMPUTADOR es (0,n), ya que un tipo de computador puede que no sea vendido o por el contrario sea vendido por varios vendedores. Además, la cardinalidad con respecto a la entidad VENDEDOR es (0,n) pues, es posible que un vendedor no haya vendido ningún tipo de computador o haya vendido varios.

En la figura 6.3 se representan dichas asociaciones, con sus respectivas cardinalidades.

Otra posible solución es haber considerado una asociación entre tres entidades: CLIENTE, VENDEDOR y COMPUTADOR. Sin embargo, una asociación de este tipo es bastante compleja y no brinda mucha información. Si se da el caso de una

C-E

asociación entre las entidades Es importante tener en cuenta que el tipo de computador cambia de acuerdo al tipo 1-E

6.4.4 D

En una relación se define una llave primaria y una llave secundaria. La llave primaria es la que define la identidad de la relación. La llave secundaria es la que define la identidad de los atributos.

Toda relación tiene una llave primaria. Una llave secundaria es una llave que no es la llave primaria.

Así, se define una llave primaria y una llave secundaria.

Llaves primarias y secundarias

Puesto que una llave primaria es una llave que define la identidad de la relación, una llave secundaria es una llave que define la identidad de los atributos.

ón pueden darle la pista al s entidades.

debe establecer:

así como sus tipos y ionario de datos.

que los clientes están interesados de crear una asociación (compra) y ésta es del tipo M-N, ya que un tadores y un tipo de computador icularse las cardinalidades de esta iente éste debe comprar al menos

De ahí que la cardinalidad con ndo lugar, un tipo de computador rario por varios clientes, entonces TADOR es (0,n).

entidades COMPUTADOR y . A esta asociación se le llamará de computador puede ser vendido er varios tipos de computadores. idad COMPUTADOR es (0,n). ya dido o por el contrario sea vendido dad con respecto a la entidad vendedor no haya vendido ningún sociaciones, con sus respectivas

a asociación entre tres entidades. Sin embargo, una asociación de este formación. Si se da el caso de una

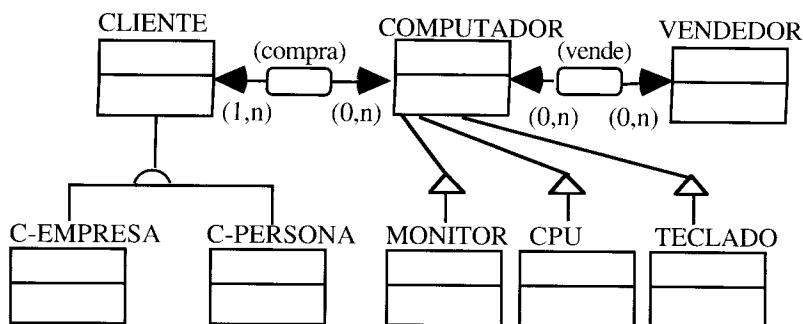


Figura 6.3 Asociaciones del esquema sobre la venta de microcomputadores

asociación entre tres o más entidades, para eliminarla se debe crear una nueva entidad y nuevas asociaciones entre las entidades originales y la entidad nueva. Es importante mencionar que si lo que interesa a la aplicación es la venta de equipos computacionales y no de los tipos de los mismos, entonces los tipos de asociación cambian. Por ejemplo, la asociación CLIENTE (compra) COMPUTADOR sería de tipo 1-N y la asociación COMPUTADOR (vende) VENDEDOR sería de tipo N-1.

6.4.4 Determinar llaves primarias y alternas

En una tercera etapa, se establecen los atributos que van a conformar las llaves primarias de cada una de las entidades. También, una asociación puede identificarse por medio de atributos y conformar asimismo una llave.

Toda entidad debe contar con una llave primaria. Si es posible, también establecer para las entidades, posibles llaves alternas con la notación de la figura 2.10.

Así, se debe recordar que los valores de los atributos que conforman una llave primaria deben ser únicos y no pueden ser nulos.

Llaves primarias en la herencia

Puesto que los objetos de una super-entidad y los de las sub-entidades respectivas representan los mismos objetos del mundo real, es natural

establecer la misma llave primaria -la de la super-entidad-, como llave primaria de todas las entidades involucradas en la herencia, sin importar el número de niveles que tenga dicha herencia.

Llaves primarias en una composición

En una composición las entidades involucradas -compuesto y componentes- tienen llaves primarias diferentes. Sin embargo, en el proceso de traducción al modelo relacional se debe crear una relación que involucre todas las llaves primarias de la composición.

Ejemplo (continuación).

En este caso se pueden definir las siguientes llaves primarias:

- Num-Cliente para la entidad CLIENTE,
- Num-producto para la entidad COMPUTADOR,
- Nombre para la entidad VENDEDOR,
- Número-teclado para el tipo de entidad TECLADO,
- Número-cpu para el tipo de entidad CPU y
- Número-monitor para el tipo de entidad MONITOR.

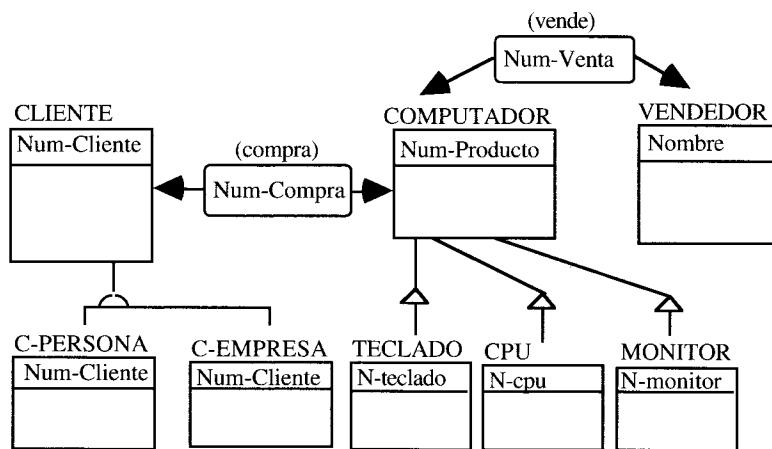
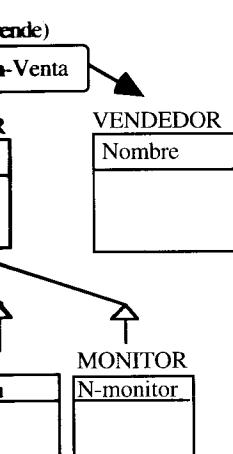


Figura 6.4 Llaves primarias y alternas del esquema sobre la venta de microcomputadores

er-entidad-, como llave
a herencia, sin importar

uercadas -compuesto y
es. Sin embargo, en el
debe crear una relación
omposición.

imarias:



venta de microcomputadores

Puesto que las entidades CLIENTE-EMPRESA y CLIENTE-PERSONA son especializaciones de la entidad CLIENTE, éstas deben tener la misma llave primaria que la super-entidad asociada, ya que representan los mismos objetos del mundo real.

Por su parte, la asociación (vende) se identifica por medio de un número. Con respecto a la asociación (compra), ésta se puede también identificar por medio de un número. En la figura 6.4 se aprecia esta situación.

6.4.5 Determinar los atributos no llave y definir los dominios de los atributos

Además de los atributos que conforman las llaves primarias y llaves alternas, se debe establecer el resto de los atributos, llamados *atributos no-llave*. Los atributos no llave sirven para caracterizar aun más cada una de las entidades.

Para cada atributo de una entidad se debe definir un dominio el cual debe contar, al menos, con los siguientes parámetros:

- tipo de dato
- longitud
- rango
- significado del atributo
- valores permitidos
- unicidad
- soporte de nulos

Ejemplo (continuación).

En el siguiente cuadro se establecen las características de algunos atributos.

Cada uno de los atributos deben ser ubicados en la relación que mejor los determine. Si se tiene una jerarquía, se debe ubicar en el nivel más alto que se pueda, pues esto facilita posteriormente el proceso de normalización.

En la figura 6.5 se puede apreciar el esquema después de la aplicación de esta etapa:

CLIENTE

Atributos	Características	Tipo
Número-Cliente	tipo de dato: longitud: rango: significado: unicidad: nulo:	INTEGER 4 dígitos 0001-9999 Número de un cliente de la Compañía sí (es llave primaria) no (es llave primaria)
Nombre	tipo de dato: longitud: rango: significado: unicidad: nulo:	CHARACTER 20 caracteres Nombre del cliente no si
Dirección	tipo de dato: longitud: rango: significado: unicidad: nulo:	CHARACTER 30 caracteres Nombre del cliente no si
Teléfono	tipo de dato: longitud: rango: significado: unicidad: nulo:	INTEGER 7 dígitos 1111111-9999999 Número de teléfono del cliente no si

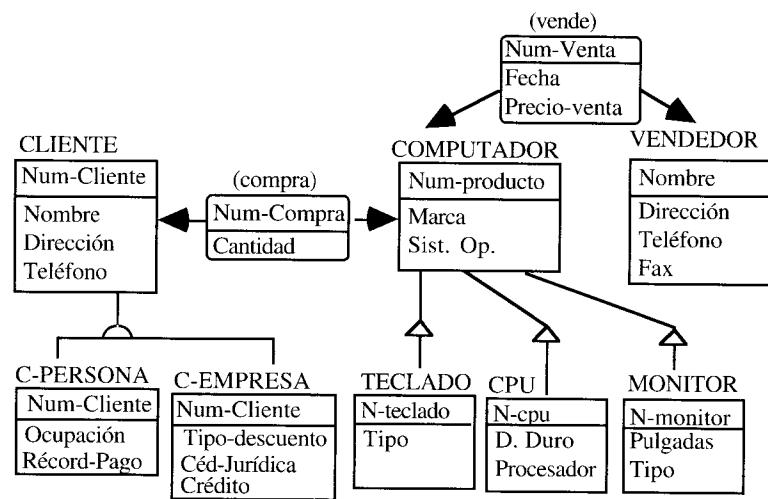


Figura 6.5 Atributos no llave del esquema sobre la venta de microcomputadores

6.4.6 Reemplazar asociaciones M-N

Es muy común, cuando se está realizando un esquema de aplicación, tener asociaciones del tipo M-N. Como se ha visto anteriormente, cada una de estas asociaciones debe sustituirse por medio de la creación de una nueva entidad y dos asociaciones 1-N. Por su parte, para esta nueva entidad se debe considerar cada una de las etapas anteriores -establecer llaves, atributos no llave, dominios, etc.-.

Ejemplo (continuación).

En este caso se tienen dos asociaciones M-N. Así, se establecen dos entidades: COMPRA y VENTA y cuatro asociaciones 1-N:

- COMPUTADOR (computador-comprado) COMPRA
- CLIENTE (compra-cliente) COMPRA
- COMPUTADOR (computador-vendido) VENTA
- VENDEDOR (venta-vendedor) VENTA

En la figura 6.6 se muestra el esquema después de aplicar la etapa referente a la transformación de las asociaciones M-N.

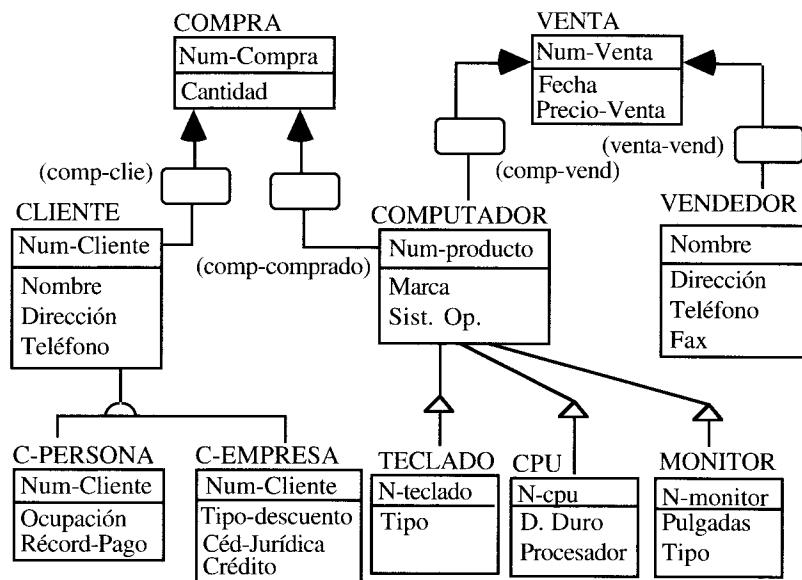


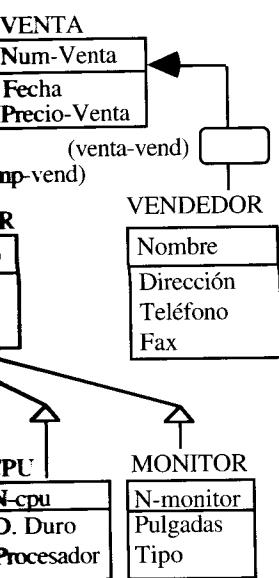
Figura 6.6 Traducción de las asociaciones M-N del esquema sobre la venta de microcomputadores

Es interesante notar que a partir de este momento se tiene un esquema de aplicación con solo asociaciones de los tipos 1-1 y 1-N, como las que se establecen en [FLEM89]. Además, se tienen estructuras de herencia y de composición, las cuales serán traducidas en su equivalente relacional en las secciones 6.5.2 y 6.5.3.

6.4.7 Determinar las llaves externas de las asociaciones

Antes de hacer la traducción del esquema de aplicación a su equivalente relacional, es importante establecer las llaves externas. Estas son necesarias pues van a traducir, en el modelo relacional, las asociaciones 1-1 y 1-N. También, se debe mencionar que las llaves

aplicar la etapa referente a la



esquema sobre la venta de

mento se tiene un esquema
os 1-1 y 1-N, como las que
en estructuras de herencia y
en su equivalente relacional

as asociaciones

esquema de aplicación a su
er las llaves externas. Estas
el modelo relacional, las
mencionar que las llaves

externas emigran de la entidad uno hacia la entidad muchos y, según el contexto, pueden formar parte de la llave de esta entidad o por el contrario, formar parte de sus atributos no llave.

Ejemplo (continuación).

Sean dos asociaciones del esquema de aplicación general que se muestran en la figura 6.7.

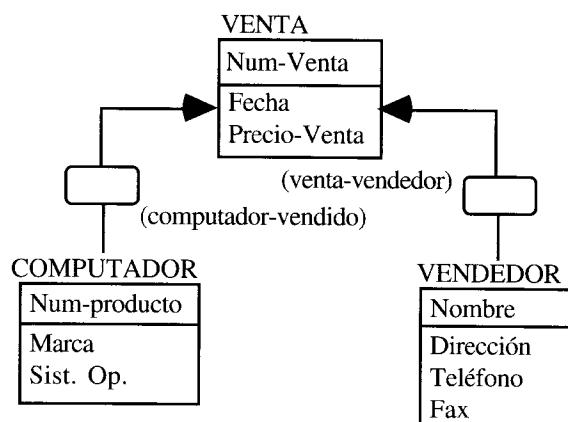


Figura 6.7 Ejemplo de dos asociaciones 1-N

En este caso la asociación (computador-vendido) se traduce ubicando la llave primaria de la entidad COMPUTADOR -Num-producto- en la entidad VENTA. Lo mismo se establece entre las entidades VENDEDOR y VENTA. La ubicación de las llaves externas, en este caso, se hace fuera de la llave primaria de la entidad VENTA. En la figura 6.8 se muestra la ubicación de las llaves externas de las asociaciones en cuestión.

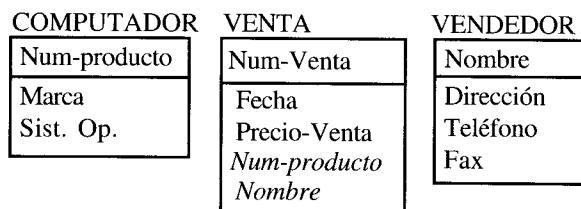


Figura 6.8 Ejemplo de llaves externas

Con respecto a una asociación 1-1, la dirección de la misma la puede establecer el analista. Entonces, la asociación se traduce colocando la llave primaria de la entidad que se escogió como fuente en la entidad que se escogió como destino. No se deben colocar ambas llaves primarias como externas.

Al aplicar este paso al ejemplo que se viene explicando, se establece el diagrama de la figura 6.9.

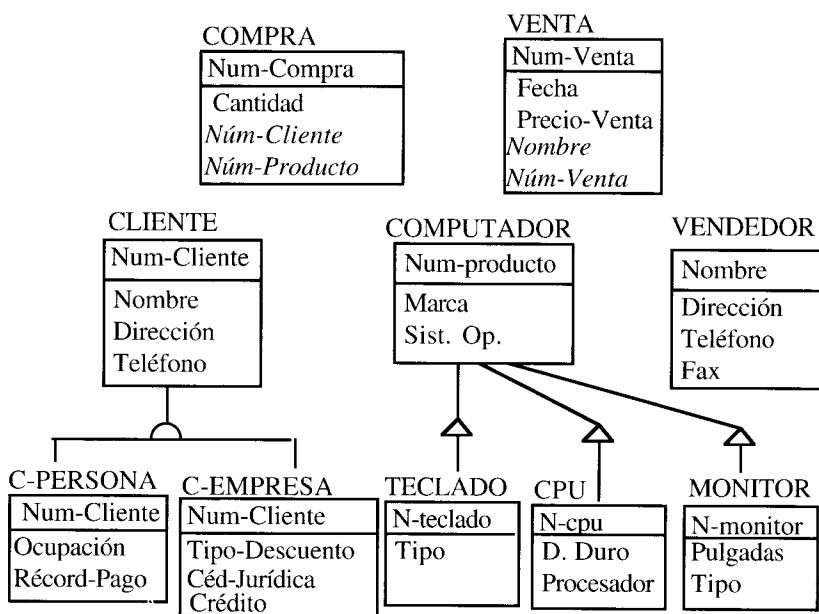


Figura 6.9 Ubicación de las llaves externas del esquema sobre la venta de microcomputadores

6.5 TRANSFORMACION AL MODELO RELACIONAL

En realidad, esta fase del diseño de la base de datos debe ser posterior a la integración de los esquemas de aplicación en un esquema conceptual global. Sin embargo, no se establecen inconsistencias, si la misma se explica en la presente sección.

Una vez que se ha establecido la función de cada entidad, se estudiará la relación entre las entidades y las integridades.

Así, para cada atributo, se establecerán las restricciones de validación.

6.5.1 Entidades

La primera etapa es establecer el esquema conceptual en forma relacional. Se deben definir las entidades y las relaciones entre ellas.

Como se ha visto, las entidades se encuentran agrupadas en super-entidades y en entidades simples.

Ejemplo

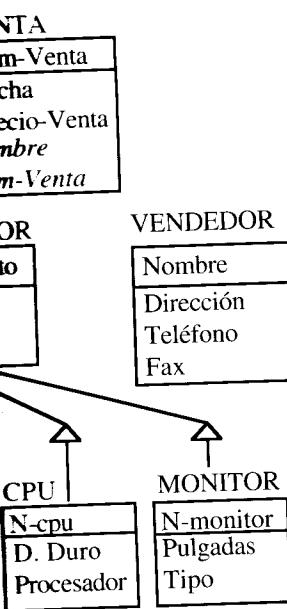
Según el esquema conceptual:

- CLIENTE
- COMPRA
- VENDEDOR
- VENTA
- COMPUTADOR

Una vez que se han establecido las entidades y las relaciones, se procederá a la transformación al modelo relacional.

ción de la misma la puede se traduce colocando la fuente en la entidad que **unir** ambas llaves primarias

de explicando, se establece



sobre la venta de microcomputadores

RELACIONAL

de datos debe ser posterior
ón en un esquema conceptual
consistencias, si la misma se

Una vez que se tenga el esquema de aplicación que traduce una función específica dentro de un sistema de información, se va a estudiar cómo pasar este esquema de aplicación a un esquema relacional, es decir, a un conjunto de relaciones y varias reglas de integridad asociadas.

Así, para cada uno de los conceptos definidos en el capítulo 2 -entidad, atributo, herencia, composición- se introduce su equivalente relacional.

6.5.1 Entidades

La primera regla que se establece es el hecho de que cada entidad del esquema de relación se debe traducir en una relación del modelo relacional. Los atributos de la entidad serán los atributos de la relación y las llaves primarias de las entidades serán las llaves primarias de las relaciones.

Como primer paso solo se consideran las entidades que no se encuentran involucradas en una herencia o una composición, excepto las super-entidades y las entidades compuestas. Las sub-entidades y las entidades componentes se analizan posteriormente.

Ejemplo (continuación).

Según el caso, se tienen las siguientes relaciones:

- **CLIENTE**(Num-Cliente, Nombre, Dirección, Teléfono)
- **COMPRA**(Num-Compra, Cantidad, Num-Cliente, Num-producto)
- **VENDEDOR**(Nombre, Dirección, Teléfono, Fax)
- **VENTA**(Num-Venta, Precio, Fecha, Nombre, Num-producto)
- **COMPUTADOR**(Num-producto, Marca, Sistema-Operativo)

Una vez que se establecen las relaciones, se procede a la creación de las mismas usando un lenguaje de consultas, como por ejemplo SQL.

Ejemplo (continuación).

Para definir las relaciones VENDEDOR, COMPUTADOR y VENTA, se puede hacer de la siguiente forma:

```
create table vendedor
  (nombre      char(30),
   direccion   char(40),
   telefono    integer,
   fax         integer,
   primary key (nombre),

create table computador
  (num_producto integer,
   marca       char(10),
   sistema_operativo char(20),
   primary key (num_producto),

create table venta
  (num_venta    integer,
   fecha_venta  date,
   nombre        char(30),
   num_producto integer,
   primary key (num_venta),
   foreign key nombre references vendedor,
   foreign key num_producto references computador);
```

En este caso se puede apreciar la dependencia de la entidad VENTA con respecto a las entidades VENDEDOR y COMPUTADOR.

6.5.2 Herencias

Con respecto a una herencia de un esquema de aplicación, se debe recordar que lo que se persigue es afinar una entidad llamada super-entidad en otras más especializadas llamadas sub-entidades, las cuales pueden brindar información más completa sobre los objetos que conforman la super-entidad. Como los objetos de la super-entidad y de las sub-entidades representan los mismos elementos del mundo real.

deben .
propios

Ejem

En la
aplica

F

El corres

Figura

ADOR y VENTA, se puede

deben contar con la misma llave primaria y cada sub-entidad con sus propios atributos.

Ejemplo (continuación).

En la figura 6.10 se muestra la herencia que se encuentra presente en el esquema de aplicación.

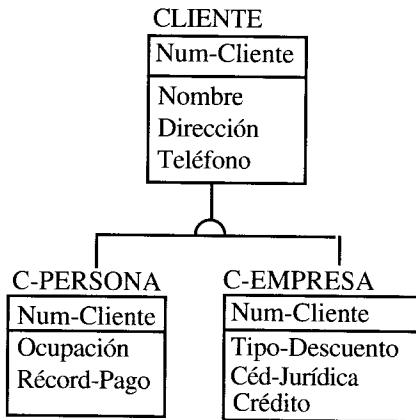


Figura 6.10 Herencia del esquema sobre la venta de microcomputadores

El correspondiente relacional de la herencia, se presenta en la figura 6.11.

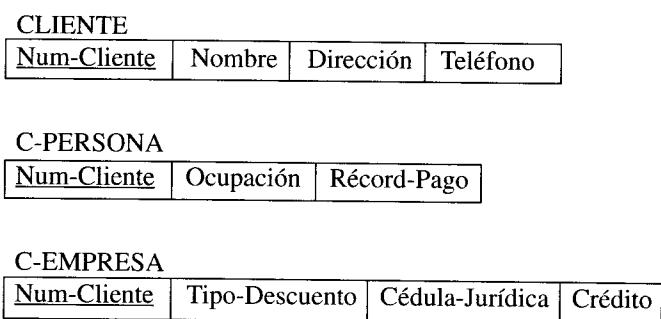


Figura 6.11 Relaciones que traducen la herencia del esquema sobre la venta de microcomputadores

Sin embargo, una vez que una herencia se traduzca en su correspondiente relacional, se deben definir adicionalmente varias reglas de integridad que permitan establecer las conexiones entre la super-entidad y las sub-entidades.

Una de estas reglas debe ser de *supresión*, la cual se aplica cuando un objeto de la super-entidad CLIENTE se suprime. En este caso, su valor asociado en alguna de las sub-entidades: C-PERSONA o C-EMPRESA, también debe suprimirse.

Por su parte, si se introduce un nuevo objeto en alguna de las sub-entidades, se debe también introducir su correspondiente valor asociado en la super-entidad.

Estas situaciones se esquematizan en la figura 6.12.

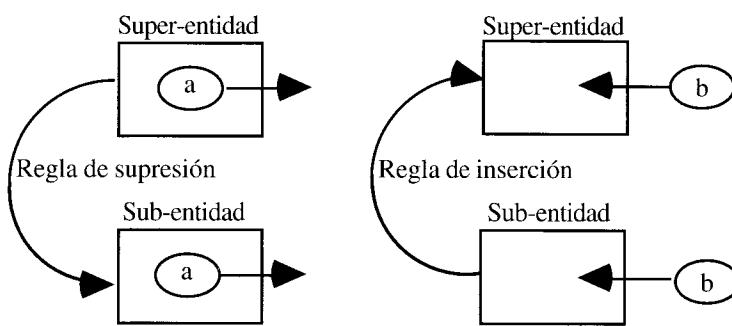


Figura 6.12 Reglas de supresión e inserción en una herencia

6.5.3 Composiciones

Con respecto a la composición, se debe recordar que traduce la asociación del tipo *es parte de*. Por esta razón, un objeto compuesto se forma a partir de objetos que se encuentran en las entidades componentes.

Ejemplo (continuación).

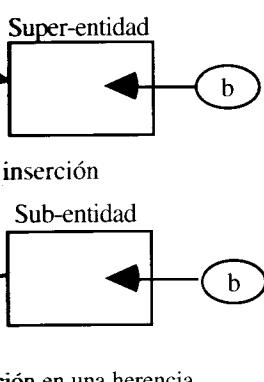
En la figura 6.13 se puede apreciar la composición presente en el esquema.

encia se traduzca en una entidad compuesta adicionalmente varias relaciones que establecen conexiones entre la super-entidad y sus componentes.

La figura 6.12 ilustra el esquema de herencia, la cual se aplica cuando una entidad compuesta tiene una relación con otra entidad que no es parte de la composición.

Un objeto en alguna de las entidades tiene un correspondiente valor asociado.

Figura 6.12.



Operación en una herencia

Se debe recordar que traduce la herencia, un objeto compuesto que se encuentra en las entidades.

Operación presente en el esquema.

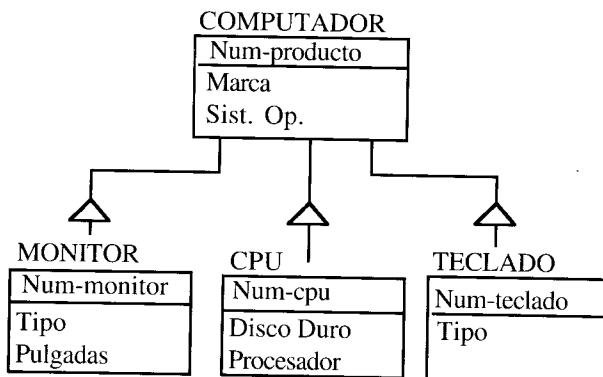


Figura 6.13 Composición del esquema sobre la venta de microcomputadores

En este caso, se puede establecer un equivalente en el modelo relacional. En efecto, la entidad compuesta se traduce en una relación con los mismos atributos que la entidad, como se hizo anteriormente. Por su parte, la misma relación se presenta con los componentes de la composición.

Además, para traducir el hecho de que una entidad se constituye en otras entidades, es necesario crear una relación, cuya llave primaria será la del compuesto y los atributos no llave serán los atributos que contienen las llaves primarias de los componentes. Esta nueva relación incluirá la información de las tuplas que se encuentran conectadas.

Si una composición tal, se pueden tener tuplas en una relación componente que no se asocian a ninguna tupla de la relación compuesta.

Ejemplo (continuación).

En la figura 6.14 se muestra cómo se realiza la traducción de la composición del esquema de aplicación en relaciones.

COMPUTADOR		
Num-producto	Marca	Sist. Operativo

MONITOR		
Num-monitor	Tipo	Pulgadas

Fig. 6.14 continúa...

CPU
Num-cpu Disco Duro Microproc.
TECLADO
Num-teclado Tipo
COMP-MON-CPU-TEC
Num-producto Num-monitor Num-cpu Num-teclado

Figura 6.14 Relaciones que traducen la composición del esquema sobre la venta de microcomputadores

Una vez que se hayan creado las relaciones respectivas, se definen varias reglas de integridad como en el caso de la herencia.

En efecto, si se desea introducir o suprimir una tupla en la relación compuesta -COMPUTADOR-, la regla debe exigir respectivamente la inserción o supresión de las tuplas correspondientes en las relaciones componentes. En la figura 6.15, se esquematiza esta situación.

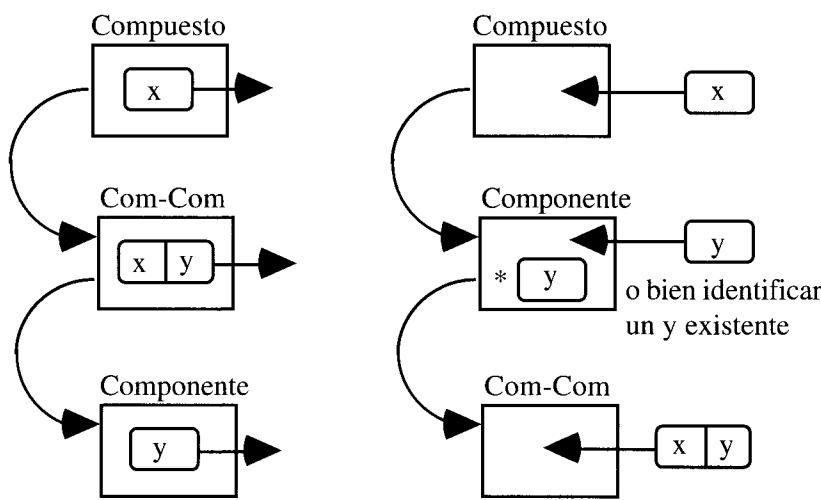


Figura 6.15 Reglas de supresión e inserción en el compuesto de una composición

Así, se suprimir relación tupla en que son tuplas ex

Por últi son perm enlace. L posibles.

Finalm reglas de visto en dependencia descomponer el fin de g propias de

Sin emb el de hor consistente rendimiento desnormaliz columnas implemen como se pu

Ejemplo:

VEND
Núm-
1234
4587
6589

Así, si se desea suprimir una tupla de la relación *Compuesto*, se deben suprimir también las tuplas asociadas en las relaciones *Componente* y la relación de enlace llamada *Com-Com*. Por su parte, si se introduce una tupla en la relación compuesto, se debe hacer lo mismo en las relaciones que son componentes y la relación de enlace o bien identificar algunas tuplas existentes en estas relaciones.

Por último, las supresiones de tuplas en las relaciones componente no son permitidas si éstas se encuentran identificadas en la relación de enlace. Las inserciones en las relaciones componentes siempre son posibles.

Finalmente, una vez que se hayan establecido las relaciones y las reglas de integridad, se procede a aplicar el proceso de normalización visto en el capítulo anterior. Para ello, se determinan posibles dependencias funcionales y multivaluadas, para aplicar el teorema de descomposición y obtener relaciones en una forma normal superior, con el fin de garantizar la eliminación de las anomalías de almacenamiento propias de formas normales inferiores.

Sin embargo, es importante mencionar que, si bien un proceso como el de normalización puede ayudar a establecer una base de datos consistente en la teoría, muy a menudo en la práctica, por razones de rendimiento, se requerirá aplicar un proceso inverso de *desnormalización controlada*. Esto es, poder duplicar o redefinir columnas o incluso redefinir relaciones sobre una base de datos implementada, con el fin de obtener un mejor rendimiento del sistema, como se puede apreciar en el siguiente ejemplo.

Ejemplo. Sean las relaciones que se muestran en la figura 6.16

VENDEDOR

Núm-Vendedor	Nombre	Dirección	Teléfono	Salario
1234	Juan Salas	Heredia	237-12344	200,000
4587	Ana Coto	Alajuela	442-4405	150,000
6589	Marvin Mora	Limón	667-0989	400,000

Fig. 6.16 continúa...

VENTA

Núm-Venta	Precio	Fecha-venta	Núm-Vendedor
3356-909	500,000	03/06/95	1234
2345-809	600,000	01/06/95	4587
2354-809	500,000	15/10/95	4587
4598-890	300,000	11/01/96	6589

Figura 6.16 Ejemplo de relaciones

En este caso, dichas relaciones se encuentran normalizadas. Por otra parte, suponer que cada una de estas relaciones contiene varias decenas de miles de tuplas y que la única conexión que se requiere entre estas relaciones es una consulta en la que se solicita información de los montos vendidos y los salarios devengados por los vendedores. Así, se debe realizar un join para asociar a cada venta, el salario de cada vendedor. Esta consulta puede ser muy cara.

Por esta razón, la persona encargada de darle mantenimiento a la base de datos podría pensar en adicionar la columna salario a la relación VENTA, según se muestra en la figura 6.17.

VENTA

Núm-Venta	Precio	Fecha-venta	Núm-Vendedor	Salario
3356-909	500,000	03/06/95	1234	200000
2345-809	600,000	01/06/95	4587	150000
2354-809	500,000	15/10/95	4587	150000
4598-890	300,000	11/01/96	6589	400000

Figura 6.17 Adición de columnas a la relación VENTA

Así, la consulta se ubicaría en una sola relación y por lo tanto se requerirían solo operadores unarios -proyecciones y selecciones-, y podría mejorarse en forma significativa el tiempo de respuesta del sistema en esta consulta.

Resumiendo, se puede decir que una metodología de diseño es una herramienta útil que permite guiar al diseñador en el proceso de desarrollo de la base de datos. Sin embargo, una misma aplicación puede tener matices diferentes, según el contexto de la empresa, así como las necesidades de la misma en información. El diseño sigue siendo más un

Iúm-Vendedor
234
587
587
589

zadas. Por otra parte, suponer
nas de miles de tuplas y que
s es una consulta en la que se
salarios devengados por los
cada venta, el salario de cada

enimiento a la base de datos
relación VENTA, según se

Vendedor	Salario
	200000
	150000
	150000
	400000

ión VENTA

lo tanto se requerirían solo
podría mejorarse en forma
ta consulta.

ología de diseño es una
ador, en el proceso de
misma aplicación puede
la empresa, así como las
eño sigue siendo más un

arte que una ingeniería. Así, el discernir lo que debe quedarse o debe irse de un diseño de bases de datos dependerá más del intelecto que de la máquina o de la herramienta.

EJERCICIOS Y PREGUNTAS DE REPASO

6.1 Explicar las tres generaciones de metodologías de desarrollo de sistemas.

6.2 Dar los nombres de seis metodologías de desarrollo de sistemas

6.3 Explicar los objetivos principales de un diccionario de datos.

6.4 ¿Qué características debe tener un diccionario de datos?

6.5 ¿Cuáles son los beneficios de un diccionario de datos?

Modelar las siguientes situaciones utilizando la metodología presentada en el presente capítulo

6.6 Se desea construir una base de datos relacional para la administración del transporte de mercadería a través del territorio centroamericano, a solicitud de una Cooperativa de Transportistas. Esta Cooperativa posee para tal fin un número de camiones de diferentes tipos: cisternas, vagonetas, trailers, tandem, etc.

Los choferes pueden conducir cualquier camión según la ciudad en que vivan. Para efectuar un envío de mercadería de una ciudad a otra, el cliente hace la solicitud de envío caracterizada por el tipo de mercadería y el peso de la misma, la ciudad de salida y la ciudad de destino. A partir de esta información y en función de los camiones disponibles, se le asigna al cliente un camión en una fecha determinada.

6.7 Se desea establecer un sistema para atender los clientes de una cadena de restaurantes de comida rápida. Una vez que el cliente llega a uno de los restaurantes, observa el menú en una pantalla, se le toma el menú y el camarero lo introduce en la computadora. Esta información llega a la caja y a la cocina del restaurante.

Posteriormente, esta información es enviada a las oficinas centrales.

- 6.8 Se desea, en una venta de discos de música, desarrollar un sistema que permita responder rápidamente a preguntas como las siguientes:

- ¿Cuáles son los álbumes de ópera?
- ¿Quién canta la canción “Idiota por tí”?
- ¿En cuáles álbumes, Pedro Vargas canta canciones de Agustín Lara?
- ¿Bajo qué marcas se producen los discos en que canta Juan Luis Guerra?
- ¿Cuántos álbumes diferentes hay de Mozart?

- 6.9 Se desea, en forma automatizada, controlar las solicitudes hechas por los diferentes clientes de la Empresa *Manzana Inc.*, especializada en la venta de equipos computacionales. Estas solicitudes son hechas cuando el cliente requiera de algún servicio técnico.

Los requerimientos del sistema son los siguientes:

- Mantener la información de las solicitudes. Se debe capturar la información que brinde el cliente una vez que éste genere una solicitud por concepto de servicio técnico. Además, mantener el estado de la solicitud como activo, mientras no sea terminada a entera satisfacción del cliente.
- Mantener la trayectoria de cada solicitud, por ejemplo: hora en que se recibió la solicitud, quién la solicitó, etc. Así, se podrá coordinar en forma eficiente la asignación de las solicitudes a los técnicos.
- Mantener información de la labor que realiza cada técnico, brindando estadísticas que le permita a la gerencia controlar y evaluar la calidad, el desempeño y el rendimiento de cada uno.

viada a las oficinas
desarrollar un sistema
preguntas como las

anciones de Agustín

en que canta Juan

art?

s solicitudes hechas por
a Inc., especializada en
solicitudes son hechas
técnico.

uientes:

es. Se debe capturar la
z que éste genere una
. Además, mantener el
ras no sea terminada a

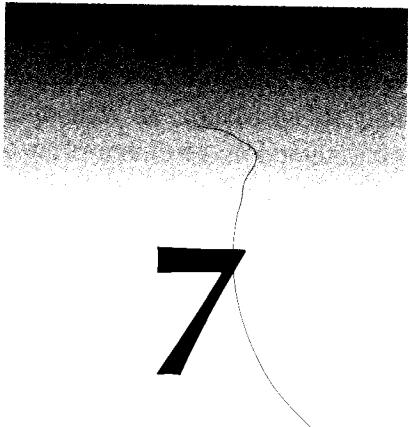
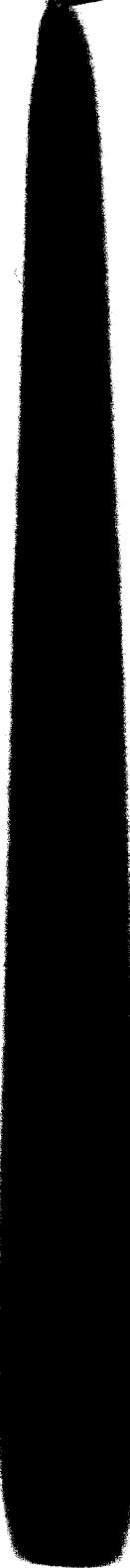
, por ejemplo: hora en
citó, etc. Así, se podrá
de las solicitudes a los

realiza cada técnico,
la gerencia controlar y
ndimiento de cada uno.

- Mantener un Histórico de todas las solicitudes hechas por los clientes para efectos de consulta, estadísticas y reportes.
- Brindar información oportuna al cliente en el caso en que deseé consultar.
- El sistema debe efectuar el proceso de facturación de los trabajos realizados a los clientes que así lo requieran, dependiendo del tiempo aplicado a la reparación del equipo y los componentes cambiados. El sistema debe contemplar las diferentes visitas que realiza un técnico al cliente por concepto de reparación de equipo.

6.10 Se considera la compañía Happy Rent a Car que alquila vehículos. Un cliente, ya sea vía telefónica o presentándose en alguna de las oficinas de la compañía, puede consultar las tarifas diarias, semanales y mensuales de los tipos de automóvil que se tienen disponibles, ya sean éstos con libre kilometraje o montos por kilómetro. Se cuenta con automóviles, vehículos de doble tracción y microbuses. Una vez que el usuario escoge la opción, se debe verificar que exista un automóvil disponible. Posteriormente, se le solicita la tarjeta de crédito y a continuación se pasa a emitir la factura respectiva. El sistema lleva un control del estado de los automóviles, cambios de aceite, kilometraje, estado de las llantas, etc.

Una vez que el cliente devuelve el vehículo se debe actualizar lo referente al estado y disponibilidad de los mismos.



7

Integración de Esquemas de Aplicación

«Si por su forma vamos a considerar la totalidad de la materia como producto de sus partes y de sus fuerzas y como su capacidad de fundirse automáticamente a otras materias que aproximan estas partes entre sí, tendremos ante nosotros un procedimiento mecánico de fusión de esas partes. De esta manera, de ningún modo se muestra ante nosotros la noción de totalidad como objetivo cuya posibilidad intrínseca supone forzosamente la idea de una totalidad de la cual dependen incluso las propiedades y el modo de actividad de las partes, tal como debemos concebir la totalidad orgánica.»

Emanuel Kant (1724-1804),
filósofo alemán.

7.1 INTRODUCCION

En el capítulo anterior se estudió una metodología de modelaje y diseño de bases de datos, que permite obtener esquemas de aplicación de una función particular, los cuales se traducen posteriormente en un conjunto de relaciones.

Una vez obtenidos los diferentes esquemas de aplicación, se debe proceder a la integración de los mismos con el fin de obtener un solo esquema conceptual integrado.

En la literatura se pueden encontrar bastantes propuestas sobre este tema: [DAYA84], [SHOVA91], [SPAC94], etc. Por ejemplo, en [BATI86] se comparan una docena de metodologías de integración de esquemas. Sin embargo, estas metodologías se puede decir que son manuales y cuando el número de esquemas de aplicación, así como su complejidad se incrementa, las mismas se pueden volver inadecuadas para resolver estos problemas.

Por otra parte, la posibilidad o no de implementar sistemas expertos que ayuden a la integración de esquemas tampoco es una idea reciente. Sin embargo, en el proceso de integración, muchas decisiones que se tomen, -por ejemplo, son o no dos tablas equivalentes, es una asociación redundante o no, etc.-, atañen a la intuición e inteligencia del diseñador. Así, es muy cuestionable pensar que se puede desarrollar un sistema experto que reemplace el trabajo del diseñador en un proceso de integración.

Sin embargo, se puede pensar en una solución intermedia, es decir, desarrollar un sistema experto que ayude al diseñador en las tareas de integración que son sencillas y directas, pero que requieren de mucho tiempo por parte del diseñador. Y, en lo que se refiere a casos delicados, que se permita un diálogo entre el diseñador y el sistema con el fin de escoger la solución más pertinente.

En el presente capítulo se estudiará una herramienta de integración de esquemas, basado en el modelo semántico introducido en el capítulo 2, el cual podría ser implementado por medio de un sistema experto. Esta herramienta se inspira en la metodología presentada por A. Cavarero y el autor en [CAVA93].

Previo a la representación esta metodología, se van a introducir los conflictos propios que se deben tomar en cuenta a la hora de integrar varios esquemas de aplicación en un solo esquema global.

7.2 CONFLICTOS DE INTEGRACION

Antes de utilizar una metodología de integración de esquemas de aplicación, es importante estudiar los posibles conflictos que se pueden generar. En la figura 7.1, se muestra una clasificación debida a Kim y Seo [KIM 91] que muestra los conflictos que se dan a la hora de integrar dos esquemas de aplicación relacionales.

<i>Comparación de relaciones</i>
Comparación de una relación con otra relación <ul style="list-style-type: none"> • Conflicto en el nombre de la relación <ul style="list-style-type: none"> - Nombres diferentes para relaciones equivalentes - Mismo nombre para relaciones diferentes • Conflicto en la estructura de la relación <ul style="list-style-type: none"> - Atributos faltantes - Atributos faltantes pero implícitos • Conflicto en las restricciones de relaciones
Comparación de un conjunto de relaciones con otro conjunto de relaciones
<i>Comparación de atributos</i>
Comparación de un atributo con otro atributo <ul style="list-style-type: none"> • Conflicto en el nombre del atributo <ul style="list-style-type: none"> - Nombres diferentes para atributos equivalentes - Mismo nombre para atributos diferentes • Conflicto de valores por defecto • Conflicto de restricciones de atributos <ul style="list-style-type: none"> - Tipos de datos - Reglas de integridad de los atributos
Comparación de un conjunto de atributos con otro conjunto de atributos.
<i>Comparación de relaciones con atributos</i>
Este tipo de conflictos se dan cuando la misma información se representa en un esquema de aplicación por medio de un conjunto de relaciones y en otro esquema por medio de un conjunto de atributos.

Figura 7.1 Clasificación de los conflictos en la integración de esquemas

Para comp...
se pueden pr...
sean EA₁ y
información ...

Bajo estas...
relación y e...
dichas relaci...
caso que una...
en el otro esq...

Otra situac...
equivalente a ...

Para comp...
introducir el s...

Ejemplo. Co...
una agencia ...
dos situacion...
se muestran a...

Esquema de c...

TURISTA(Núm...
SITIO(Núm-S...
VIAJE(Núm-...

Esquema de a...

PASAJERO(Id-...
LUGAR (Id-L...
VIAJE(Id-Via...
GUIA(Id-Guía...

Si se dese...
resolver vario...
En primer lu...
equivalentes y ...

se van a introducir los
ta a la hora de integrar
ma global.

ación de esquemas de
conflictos que se pueden
cación debida a Kim y
dan a la hora de integrar

valentes
es

conjunto de relaciones

valentes
s

conjunto de atributos.

nación se representa en
de relaciones y en otro

gración de esquemas

Para comprender mejor esta clasificación y explicar los conflictos que se pueden presentar cuando se desea integrar esquemas de aplicación, sean EA_1 y EA_2 dos esquemas de aplicación e I representa cierta información que se encuentra en los dos esquemas de aplicación.

Bajo estas circunstancias, I se puede representar en EA_1 como una relación y en EA_2 como otra relación. En este caso, los nombres de dichas relaciones pueden o no ser los mismos. También puede darse el caso que una relación en un esquema sea equivalente a varias relaciones en el otro esquema de aplicación.

Otra situación es cuando un atributo en un esquema de aplicación sea equivalente a otro atributo o varios atributos o incluso una relación.

Para comprender mejor el problema de los conflictos, se va a introducir el siguiente ejemplo.

Ejemplo. Con respecto a la base de datos sobre Ecoturismo, se va a suponer que una agencia de viajes también organiza viajes a lugares de interés ecológico. Las dos situaciones se pueden representar entonces por los esquemas de aplicación que se muestran a continuación:

Esquema de aplicación 1: Club de Ecoturismo

TURISTA(Núm-Turista, Nombre-Turista, Calle, Código, Provincia, País)

SITIO(Núm-Sitio, Nombre-Sitio, Continente, Tipo)

VIAJE(Núm-Viaje, Núm-Turista, Núm-Sitio, Fech-Sal, Fech-Lleg, Ciudad-Lleg)

Esquema de aplicación 2: Agencia de Viajes

PASAJERO(Id-Pasajero, Nombre, Dirección, Teléfono)

LUGAR (Id-Lugar, Nombre, Ubicación)

VIAJE(Id-Viaje, Id-Pasajero, Id-Lugar, Ciudad-Salida, Fecha)

GUIA(Id-Guía, Nombre-Guía, Dirección-Guía, Teléfono-Guía)

Si se desean integrar estos dos esquemas de aplicación en uno solo, se deben resolver varios conflictos.

En primer lugar, en apariencia las relaciones TURISTA y VIAJERO son equivalentes y por lo tanto, deberían fusionarse como una sola relación. Este es un

conflicto entre relaciones, en donde se tienen nombres diferentes para dos relaciones que son semánticamente equivalentes. Este problema se denomina *sinonimia*. El nombre que se deja para la nueva relación fusionada depende de cuál de los nombres de las relaciones originales es el más aceptado en el contexto del nuevo esquema integrado. Además, las relaciones SITIO y LUGAR se pueden considerar equivalentes, así como las relaciones VIAJE de los dos esquemas de aplicación.

Una vez que se integren las diferentes relaciones, se debe verificar si los atributos de las relaciones integradas, son o no equivalentes. En este caso, un conjunto de atributos puede ser equivalente a otro conjunto de atributos. Así, por ejemplo, los atributos: Calle, Código Provincia y País se pueden considerar equivalentes con el atributo Dirección. Este es un conflicto típico entre dos conjuntos de atributos, los cuales se pueden ver como semánticamente equivalentes.

A continuación se muestra un posible esquema que resulta de la integración de los dos esquemas de aplicación anteriores.

TURISTA(Núm-Turista, Nombre-Turista, Dirección, Teléfono)

LUGAR(Id-Lugar, Nombre, Ubicación, Continente, Tipo)

VIAJE(Id-Viaje, Núm-Turista, Id-Lugar, Fech-Sal, Fech-Lleg, Ciudad-Lleg)

GUIA(Id-Guía, Nombre-Guía, Dirección-Guía, Teléfono-Guía)

Opuesto a la sinonimia se tiene la *homonimia*. Esta situación se da cuando dos relaciones tienen el mismo nombre, pero significan cosas diferentes. Podría ser el caso de dos relaciones que tengan el nombre de FECHA, en la cual una representa el pago de los empleados de una empresa y la otra, la fecha de recepción de mercaderías.

Con respecto a la comparación de una relación con varias relaciones, se pueden considerar dos esquemas de aplicación de un sistema de información universitario en donde la relación EMPLEADO en un esquema es equivalente a una herencia en el otro esquema de aplicación, con FUNCIONARIO como super-entidad y ADMINISTRATIVO, PROFESOR e INVESTIGADOR como sub-entidades.

Adicional a los conflictos vistos anteriormente, la metodología de integración que se presenta permite asimismo el análisis y comparación de las asociaciones, herencias y composiciones involucradas en los esquemas de aplicación considerados en el proceso de integración.

7.3

D

La

de n

aprecia

Esto sig
obtiene un
integra co
integrado a
aplicación.

Si se usa
un peso o p
el siguiente

mbres diferentes para dos
ste problema se denomina
n fusionada depende de cuál
aceptado en el contexto del
ITIO y LUGAR se pueden
JE de los dos esquemas de

debe verificar si los atributos
en este caso, un conjunto de
atributos. Así, por ejemplo, los
considerar equivalentes con el
os conjuntos de atributos, los
tes.

sulta de la integración de los

Teléfono)
ípo)
ch-Lleg, Ciudad-Lleg)
ono-Guía)

situación se da cuando dos
osas diferentes. Podría ser el
HA, en la cual una representa
ra, la fecha de recepción de

varias relaciones, se pueden
n sistema de información
un esquema es equivalente a
UNCIÓNARIO como super-
NVESTIGADOR como sub-

ente, la metodología de
el análisis y comparación
nes involucradas en los
ceso de integración.

7.3 METODOLOGIA DE INTEGRACION DE ESQUEMAS DE APLICACION

La metodología que se presenta en esta sección supone la integración de n esquemas de aplicación, los cuales se integrarán dos a dos, según se aprecia en la figura 7.2.

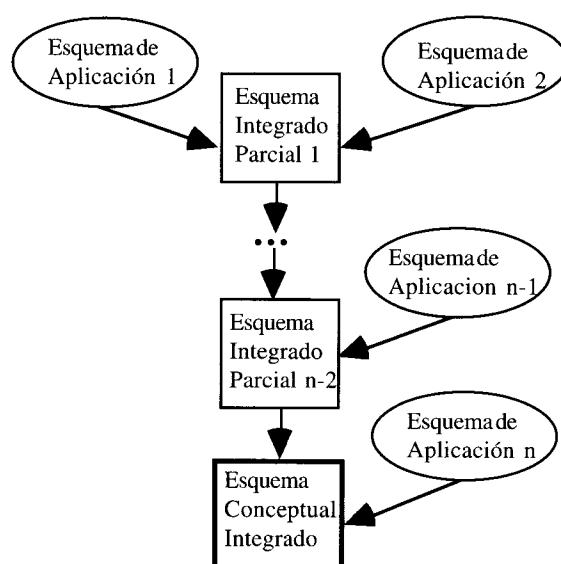


Figura. 7.2 Proceso de integración

Esto significa que, de la integración de dos esquemas de aplicación se obtiene un esquema de aplicación integrado parcial, el cual a su vez se integra con un tercer esquema de aplicación, para obtener otro esquema integrado parcial y así sucesivamente, hasta integrar los n esquemas de aplicación. De esta forma, se obtiene un esquema conceptual integrado.

Si se usa este enfoque de integrar dos esquemas a la vez, se debe dar un *peso* o prioridad a cada esquema de aplicación, con el fin de definir el siguiente orden parcial:

$$EA_1 \angle EA_2 \angle \dots \angle EA_n$$

Este peso puede determinarse por medio de varios parámetros, por ejemplo, fecha de construcción del esquema, prioridad o jerarquía del diseñador en la empresa, relevancia del esquema de aplicación en el contexto de la empresa, etc.

Así, se considera que el esquema de aplicación #1 es el de mayor peso y por ende, el primero que debe tomarse en cuenta en el proceso de integración.

La metodología de integración de esquemas se compone de las siguientes cuatro etapas:

- integración de las entidades
- integración de los enlaces
 - integración de las asociaciones
 - integración de las herencias
 - integración de las composiciones
- análisis de la equivalencia entre atributos inter-entidades
- obtención de un esquema integrado.

Cuando se aplique esta metodología a n esquemas de aplicación, cada concepto -atributo, entidad, asociación, herencia y composición- en un esquema de aplicación tendrá una etiqueta de pertenencia a dicho esquema. De esta forma, los distintos conceptos de cada uno de los diferentes esquemas de aplicación son identificables.

A continuación, se presenta cada una de las etapas de la metodología.

7.3.1 Etapa 1: Integración de las entidades

El primer paso de la metodología es la comparación de las entidades presentes en los diferentes esquemas de aplicación, con el fin de valorar si es o no pertinente la fusión de las mismas en una sola entidad.

Sean X y Y dos entidades, las cuales pertenecen a dos esquemas de aplicación EA₁ y EA₂ respectivamente. Se desea saber si estas entidades

son c
objet

La
semá
entida
comp
alterna

Al

1. X

•

• s
e

2. X

• S
e
at
or

• Si
de

Si es e
entonces
dados com

Ejempl
llamadas

varios parámetros, por
oridad o jerarquía del
a de aplicación en el

#1 es el de mayor peso
enta en el proceso de

s se compone de las

er-entidades

mas de aplicación, cada
y composición- en un
e pertenencia a dicho
os de cada uno de los
oles.

apas de la metodología.

uración de las entidades
ón, con el fin de valorar
una sola entidad.

ecen a dos esquemas de
saber si estas entidades

son o no comparables, es decir, si representan o no el mismo conjunto de objetos del mundo real.

La comparación de dos entidades se puede establecer analizando su semántica y/o valorando las llaves primarias respectivas. Cuando dos entidades tienen la misma llave primaria, es casi seguro que sean comparables, también cuando la llave primaria de una entidad es llave alterna de la otra y viceversa.

Al comparar entidades, dos casos son posibles:

1. X y Y no representan la misma entidad.

- Si el nombre de X es igual al nombre de Y, entonces se debe cambiar el nombre de una de las entidades. Las dos entidades deben aparecer en el esquema parcial integrado. Documentar estos cambios en el diccionario de datos.
- Si los nombres no coinciden, dejar las dos entidades originales en el esquema integrado parcial.

2. X y Y representan la misma entidad.

- Si el nombre de la entidad X es igual al nombre de la entidad Y, entonces colocar los atributos de la entidad Y junto con los atributos de la entidad X en una sola entidad con el nombre original.
- Si los nombres de las entidades no coinciden, buscar un nombre de consenso y agrupar los atributos en esa nueva entidad.

Si es el nombre X el que se deja en el esquema integrado parcial, entonces el nombre Y puede quedar documentado en el diccionario de datos como una entidad equivalente a la entidad X.

Ejemplo. Considerar dos entidades en dos esquemas de aplicación diferentes, llamadas CLIENTE y COMPRADOR, según se aprecia en la figura 7.3.

CLIENTE	COMPRADOR
Número-Cliente	Cédula-Jurídica
Nombre-Cliente	Nombre
Teléfono	Dirección

Figura 7.3 Ejemplo de dos entidades equivalentes

Si se desea realizar la integración de estos dos esquemas, se puede considerar que estas dos entidades son equivalentes y deberían fusionarse. Así, se debe tomar la decisión de dejar uno de los dos nombres en el esquema integrado y tal vez el otro en el diccionario de datos, como sinónimo de esta entidad.

Por otra parte suponer que es el nombre CLIENTE el que se establece como el concepto unificador, entonces siguiendo esta primera etapa, se depositan los atributos de la entidad COMPRADOR en la entidad CLIENTE, según se aprecia en la figura 7.4.

CLIENTE
Número-Cliente
Nombre-Cliente Teléfono
Cédula-Jurídica [1]
Nombre
Dirección
Crédito-Disponible

Figura 7.4 Ejemplo de entidad fusionada

En este caso, si la llave primaria de la entidad que desaparece no coincide con la llave primaria de la entidad que permanece, entonces se marca como llave alterna en esta entidad integrada.

7.3.2 Etapa 2: Integración de enlaces

Con respecto a la integración de enlaces -asociaciones, herencias y composiciones-, se pueden dar diferentes casos, según el tipo de enlace y de las características de las entidades, puesto que una integración de enlaces puede inducir a alteraciones en las entidades concernidas. Así, cada paso debe considerarse por separado.

DOR

rídica'

ponible

ivalentes

nas, se puede considerar que
onarse. Así, se debe tomar la
ma integrado y tal vez el otro
tidad.

el que se establece como el
apa. se depositan los atributos
según se aprecia en la figura 7.4.

onada

desaparece no coincide con la
s se marca como llave alterna

asociaciones, herencias y
os, según el tipo de enlace
to que una integración de
ntidades concernidas. Así

Paso 2.1 Integración de las asociaciones

En el capítulo anterior se estableció que las asociaciones podían reducirse a solo asociaciones de los tipos 1-1 y 1-N. Así, solo se consideran estos casos.

Sean las siguientes asociaciones con sus respectivas entidades:

$$\begin{array}{l} E \text{ (a) } F \\ G \text{ (b) } H \end{array}$$

Dependiendo de si las asociaciones tienen o no el mismo significado o que las entidades sean o no equivalentes, se pueden tener 5 casos, según se aprecia en la figura 7.5.

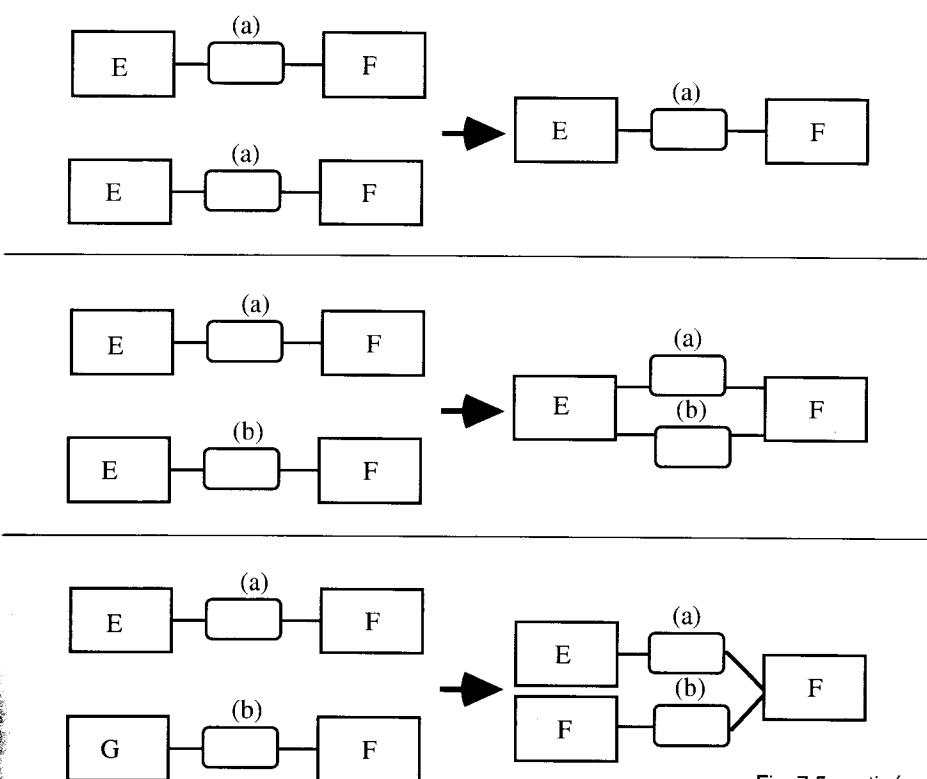


Fig. 7.5 continúa...

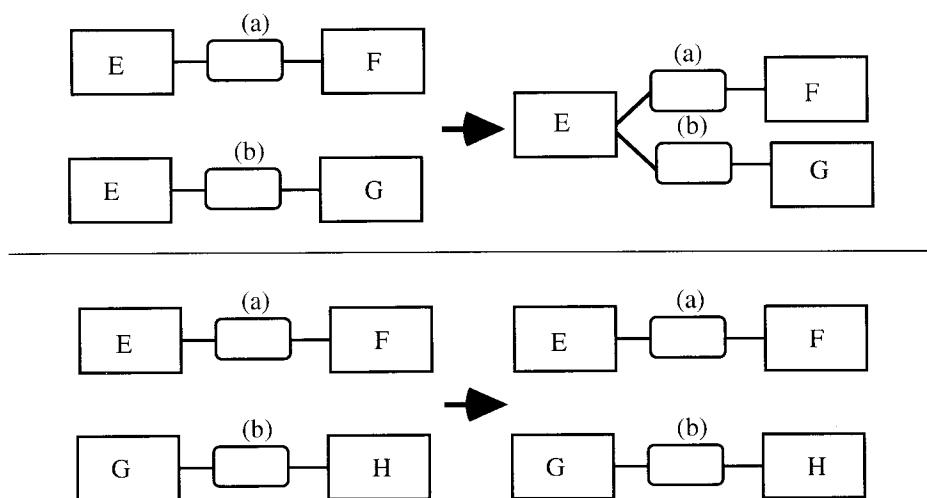


Figura 7.5 Posibles formas de integrar asociaciones

Junto a esta clasificación de posibles asociaciones entre entidades, existen casos puntuales que deben ser estudiados por separado.

En primer lugar, se debe mencionar que al integrar dos asociaciones 1-N, se puede obtener una asociación del tipo M-N; en cuyo caso, se debe crear una nueva entidad y dos asociaciones 1-N, como se estudió en el capítulo anterior.

Ejemplo. Sean dos asociaciones en dos esquemas de aplicación diferentes, según se aprecia en la figura 7.6.

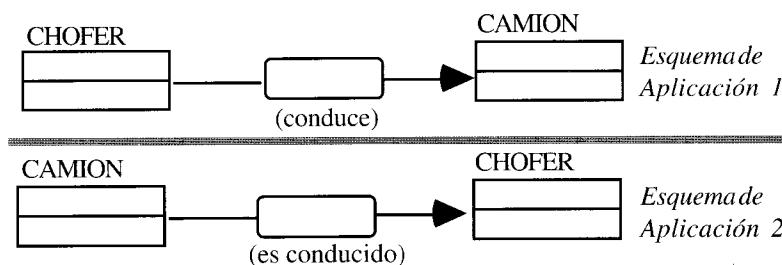


Figura 7.6 Ejemplo de asociaciones en dos esquemas de aplicación.

En este caso...
entonces...

Otra po...
que aparez...

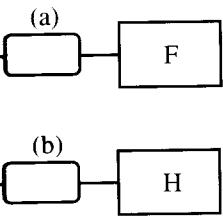
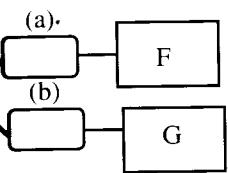
Ejemplo...

DEPA...

DIREC...

F...

En este caso...
puede est...
DEPARTA...
aplicación ...



aciones

ciones entre entidades.
por separado.

integrar dos asociaciones
M-N; en cuyo caso, se
l-N, como se estudió en

aplicación diferentes, según

N
Esquema de
Aplicación 1

R
Esquema de
Aplicación 2

as de aplicación.

En este caso la integración de esquemas dará una asociación M-N y se procede entonces a crear una nueva entidad, según se aprecia en la figura 7.7.

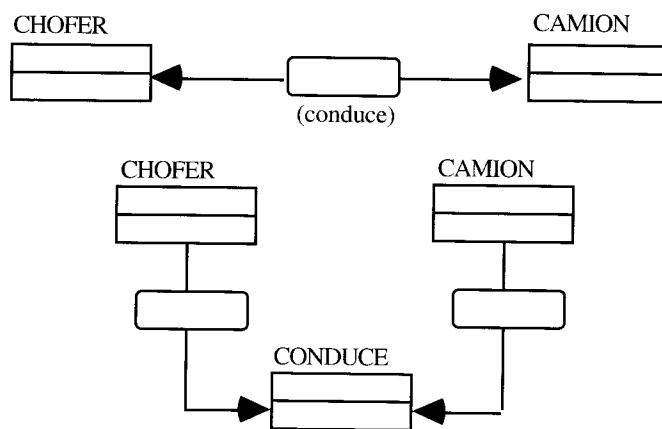


Figura 7.7 Ejemplo de asociaciones integradas

Otra posibilidad que se puede dar a la hora de integrar esquemas es que aparezcan nuevas asociaciones, y que otras se vuelvan redundantes.

Ejemplo. Sean las dos asociaciones que aparecen en la figura 7.8.

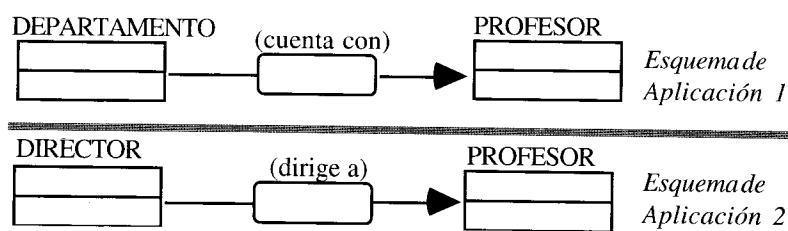


Figura 7.8 Ejemplo de asociaciones en dos esquemas de aplicación.

En este caso, al integrar los dos esquemas de aplicación, se observa que se puede establecer una asociación entre las entidades DIRECTOR y DEPARTAMENTO, la cual no existe al estar los dos esquemas de aplicación separados. Por otra parte, una vez establecida esta nueva

asociación, la asociación DIRECTOR (dirige) PROFESOR se vuelve una asociación redundante. Así, la integración de estos dos esquemas se presenta en la figura 7.9.

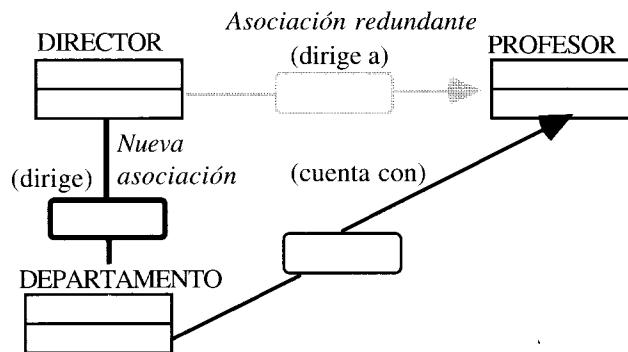


Figura 7.9 Ejemplo de asociaciones implícitas y redundantes

Es importante también mencionar que a la hora de integrar asociaciones, se deben analizar las cardinalidades mínimas y máximas y determinar las que mejor se adapten al nuevo esquema integrado.

Paso 2.2 Integración de las herencias

Cuando las entidades que son candidatas a la integración aparecen en diferentes estructuras de herencia, se puede resolver el problema por medio de la definición de un álgebra sobre esta estructura.

En primer lugar, cuando una entidad A es super-entidad de una entidad B -lo que significa que B es sub-entidad de A-, se utilizará la siguiente notación:

$$A \gg B$$

Por otra parte, si se considera un camino en una herencia desde una entidad A_1 hasta una entidad A_n , se utilizará la siguiente notación:

$$[A_1 \gg A_2 \gg A_3 \gg \dots \gg A_n]$$

FESOR se vuelve una
tos dos esquemas se

ROFESOR



redundantes

la hora de integrar
es mínimas y máximas
sistema integrado.

integración aparecen en
someter el problema por
estructura.

super-entidad de una
ad de A-, se utilizará la

una herencia desde una
guiente notación:

En este caso A_1 es super-entidad, A_2 es sub-entidad de A_1 , pero es super-entidad de la entidad A_3 y así sucesivamente. Con esta notación, se puede definir un álgebra de herencias. En efecto, sean dos herencias parciales $[A_1 \gg \dots \gg A_i \gg \dots \gg A_n]$ y $[B_1 \gg \dots \gg B_j \gg \dots \gg B_m]$ en dos esquemas de aplicación diferentes, en donde las entidades A_i y B_j son equivalentes lo que se denota por $A_i \equiv B_j$. Se puede definir la integración de herencias de la siguiente forma:

$$[A_1 \gg \dots \gg A_i \gg \dots \gg A_n] \oplus [B_1 \gg \dots \gg B_j \gg \dots \gg B_m] = \\ [[A_1 \gg \dots \gg A_{i-1}] \vee [B_1 \gg \dots \gg B_{j-1}] \gg A_i \gg [A_{i+1} \gg \dots \gg A_n] \vee [B_{j+1} \gg \dots \gg B_m]]$$

según se aprecia en la figura 7.10.

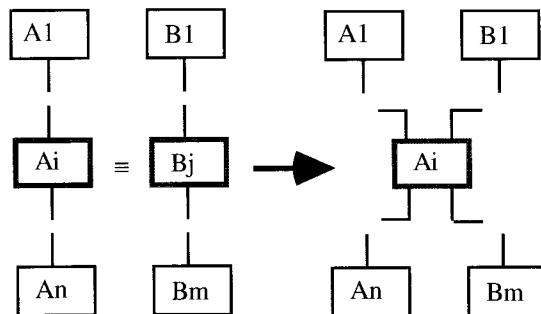


Figura 7.10 Integración de herencias

Ejemplo. Considerar las dos generalizaciones parciales siguientes:

[Medios de comunicación » TV » Film » [Comedia] \vee [Ciencia Ficción]]; y
[Actividades de esparcimiento » Film].

según se aprecia en la figura 7.11.

La integración de la entidad Film daría la siguiente generalización:

[Actividades de esparcimiento \vee [Medios de comunicación » TV] » Film » [Comedia] \vee [Ciencia Ficción]],

como se aprecia en la figura 7.12.

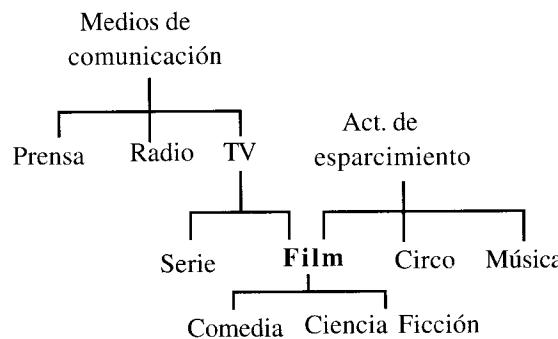
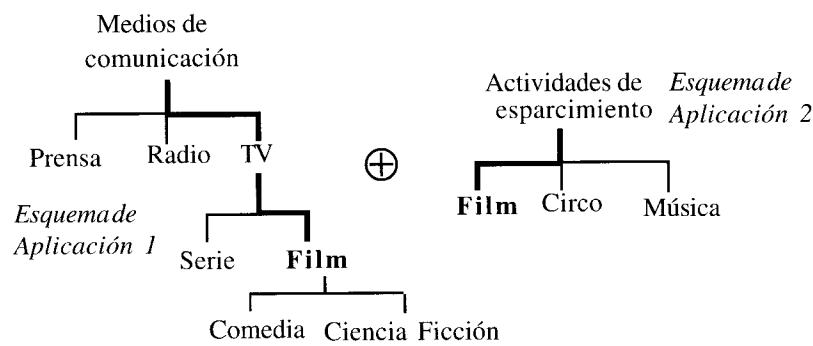


Figura 7.12 Ejemplo de integración de herencias

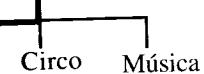
Es interesante mencionar que a partir de dos herencias parciales que se integran pueden surgir herencias múltiples, en las cuales una entidad puede tener varias super-entidades asociadas.

Paso 2.3 Integración de composiciones

De la misma forma que se hizo con las herencias, se puede establecer un álgebra sobre las composiciones. En efecto, sea una entidad A compuesto y B una entidad componente, esta situación se denota como

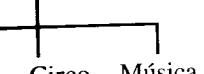
$$A \approx B$$

vidades de Esquema de
parcimient Aplicación 2



nas de aplicación diferentes

de
recimiento



Ficción

ón de herencias

e dos herencias parciales que
les. en las cuales una entidad
as.

herencias, se puede establecer
n efecto, sea una entidad A
esta situación se denota como

Entonces, sean dos composiciones parciales $[A_1 \approx \dots \approx A_i \approx \dots \approx A_n]$ y $[B_1 \approx \dots \approx B_j \approx \dots \approx B_m]$ en dos esquemas de aplicación diferentes. Además, considerar las entidades A_i y B_j como equivalentes. Se puede definir entonces la integración de composiciones de la siguiente manera:

$$[A_1 \approx \dots \approx A_i \approx \dots \approx A_n] \otimes [B_1 \approx \dots \approx B_j \approx \dots \approx B_m] = \\ [[A_1 \approx \dots \approx A_{i-1}] \vee [B_1 \approx \dots \approx B_{j-1}] \approx A_i \approx [A_{i+1} \approx \dots \approx A_n] \vee [B_{j+1} \approx \dots \approx B_m]]$$

Ejemplo. Sean $[Casa \approx Muro]$ y $[Cerca \approx Muro \approx [Cemento] \vee [Ladrillo]]$ dos composiciones parciales, según se aprecia en la figura 7.13.

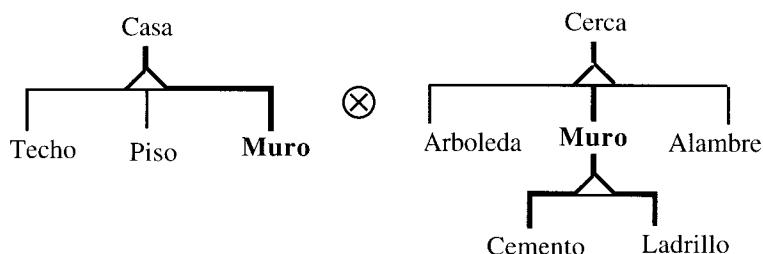


Figura 7.13 Ejemplo de dos composiciones en esquemas de aplicación diferentes

La integración de la entidad MURO daría la siguiente composición:
 $[[Casa] \vee [Cerca] \approx Muro \approx [Cemento] \vee [Ladrillo]]$,
según se aprecia en la figura 7.14.

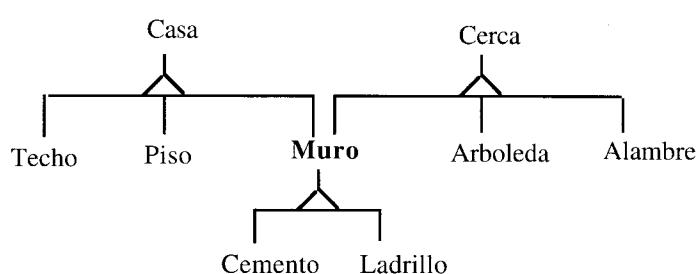


Figura 7.14 Ejemplo de integración de composiciones

7.3.3 Etapa 3: Equivalencia de los atributos inter-entidades

Al terminar con la segunda etapa, se pueden tener entidades con atributos redundantes y/o inconsistentes.

A continuación, se van a enumerar los diferentes casos que se pueden presentar y se darán las acciones requeridas para resolver el conflicto.

Sean $a_1, \dots, a_n, b_1, \dots, b_m$ los atributos de la entidad integrada en donde a_1, \dots, a_n son los atributos originales de la entidad X en un esquema de aplicación y b_1, \dots, b_m son los atributos originales de la entidad Y en otro esquema de aplicación. Sean $a \in \{a_1, a_2, \dots, a_n\}$ y $b \in \{b_1, \dots, b_m\}$ dos atributos. Entonces

1. Si a y b tienen el mismo nombre e igual significado, entonces suprimir el atributo b.
2. Si a y b tienen el mismo nombre, pero diferente significado, entonces cambiar el nombre de uno de ellos y dejar los dos.
3. Si a y b tienen diferente nombre pero con igual significado, entonces solo dejar uno de los atributos.
4. Si a y b tienen diferente nombre y diferente significado, entonces se deben dejar los dos atributos.
5. En general, si el significado de un grupo de atributos a_{i_1}, \dots, a_{i_k} es el mismo que el significado de un grupo de atributos b_{j_1}, \dots, b_{j_s} (Es el caso por ejemplo de $a_1 =$ dirección, $a_2 =$ país y $b_1 =$ calle, $b_2 =$ código, $b_3 =$ provincia, $b_4 =$ país), entonces escoger alguno de los dos grupos y suprimir el otro.

7.3.4 Etapa 4: Obtención de un esquema integrado

En esta etapa el diseñador cuenta con un conjunto de clases resultantes de las diferentes etapas de integración. Así, él puede eventualmente completar sus entidades adjuntando otros objetos.

No se debe olvidar que todos los cambios que surjan de una integración de esquemas, deben ser documentados en el diccionario de datos.

U
integ
semá
a una
del ca

EJER

7.1

7.2

7.3

7.4

7.5

E
in
ci
P
pr
T

inter-entidades

en tener entidades con
estos casos que se pueden
resolver el conflicto.

idad integrada en donde
ad X en un esquema de
s de la entidad Y en otro
} y $b \in \{b_1, \dots, b_m\}$ dos

al significado, entonces

o diferente significado.
los y dejar los dos.

con igual significado.

nte significado, entonces

o de atributos a_{i1}, \dots, a_{ik} es
de atributos b_{j1}, \dots, b_{js} (Es
= país y b_1 = calle, b_2 =
es escoger alguno de los

integrado

un conjunto de clases
egración. Así, él pue
ndo otros objetos.

e surjan de una integració
ccionario de datos.

Una vez que se cumplen las diferentes etapas del proceso de integración, se obtiene un solo esquema conceptual en el modelo semántico establecido en el capítulo 2. A partir de este momento, se pasa a una traducción al modelo relacional, según se estudió en la sección 6.5, del capítulo dedicado a la introducción de la metodología de diseño.

EJERCICIOS Y PREGUNTAS DE REPASO

- 7.1 ¿Por qué es importante contar con una herramienta que permita la integración de esquemas de aplicación?
- 7.2 Explicar por qué cualquier proceso de integración de esquemas es una tarea muy difícil.
- 7.3 Explicar cada una de las 4 etapas de la metodología de integración.
- 7.4 Dar un ejemplo de dos esquemas de aplicación en donde surjan asociaciones redundantes y asociaciones implícitas.
- 7.5 En la figura 7.15 aparecen dos esquemas de aplicación incompletos. Completar dichos esquemas estableciendo, según su criterio, las llaves, los atributos no llave y las cardinalidades. Posteriormente, aplicar el método de integración visto en el presente capítulo.

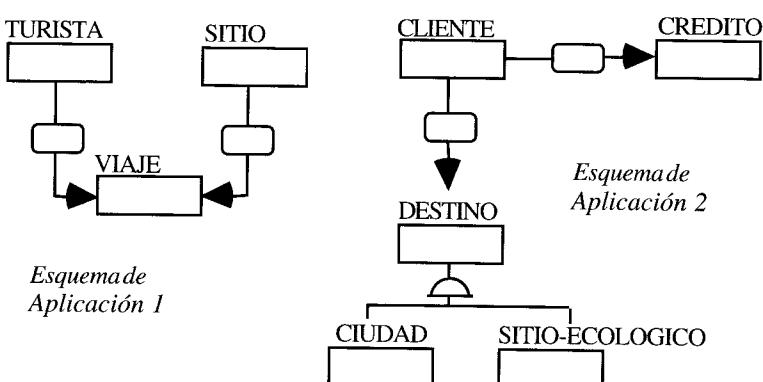
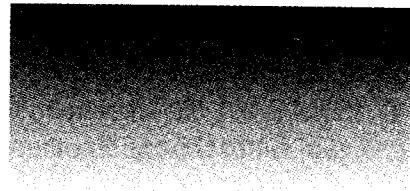


Figura 7.15 Ejemplo de dos esquemas de aplicación.



8

Evaluación y Optimización de Consultas

«Igual que un mismo artesano puede hacer dos relojes que, aun cuando ambos señalen con igual exactitud el tiempo y no haya entre ellos diferencia exterior aparente, no tienen nada semejante en la composición de sus ruedas; así, el Supremo Hacedor de las cosas tiene, sin duda, una infinidad de medios diversos a su disposición, para cada uno de los cuales pudo haber hecho que todas las cosas de este mundo aparezcan como nosotros las vemos, sin que le sea posible a la mente humana conocer cuál de todos esos medios decidió emplear.»

René Descartes (1596-1650),
matemático y filósofo francés.

8.1 INTRODUCCION

Un campo de investigación de la mayor relevancia en la tecnología de bases de datos es la referente a la implementación de los SABDs. Y dentro de la implementación, el procesamiento de las consultas juega un papel fundamental.

Así, cuando un usuario hace una consulta a una base de datos, el SABD cuenta con una serie de mecanismos que permiten el procesamiento de la misma, por medio de un acceso eficiente a los datos, es decir, en el menor tiempo posible y en forma transparente a dicho usuario.

El *procesamiento de una consulta* se ubica entre la solicitud que hace el usuario por medio de un lenguaje de consultas y los archivos en donde se encuentran almacenadas las relaciones que conforman la base de datos. Dicho procesamiento se puede a su vez descomponer en dos grandes funciones: la *evaluación* de la consulta y la *optimización* de la misma.

Cuando un usuario efectúa una consulta a la base de datos, el SABD verifica que la consulta contenga referencias válidas a las tuplas de la base de datos y seguidamente la traduce en un árbol semántico, en donde los nodos representan los operadores relacionales y las hojas representan las relaciones y vistas involucradas en dicha consulta.

Posteriormente, este árbol se pasa a un módulo llamado *optimizador de consultas* que se encarga de escoger, entre un número importante de posibilidades, una en donde el orden en que se van a ejecutar los cálculos garantiza el acceso más rápido.

Por su parte, la salida del optimizador es un *plan de ejecución* -plan de evaluación, QEP, o simplemente plan [GRAE93]- de la consulta del usuario. Luego, se genera un código para ejecutar dicho plan. El módulo que hace esto posible se denomina *cálculo de expresiones y operadores*.

Finalmente, se cuenta con un módulo de *procedimientos de acceso* que garantizan el acceso a las tuplas de la base de datos que fueron solicitadas.

En la figura 8.1 se muestra un diagrama con los diferentes pasos que sigue un SABD con el fin de evaluar y optimizar una consulta. Estos pasos y módulos se presentan a modo de referencia, pues los mismos difieren de un SABD al otro.

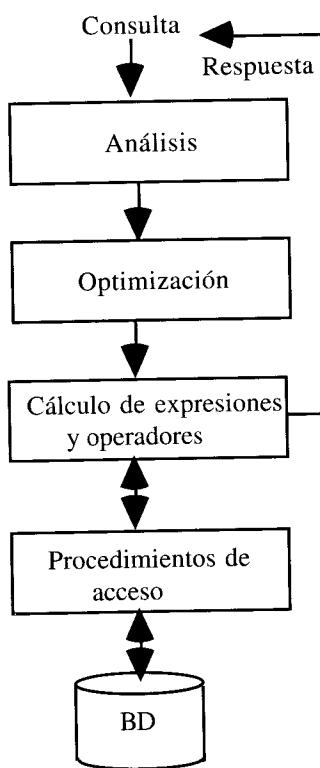


Figura 8.1 Pasos que sigue un SABD para responder a una consulta

Por su parte, cuando se habla de optimización de una consulta, se involucra una serie de técnicas y estrategias que van desde transformaciones lógicas de las consultas hasta la optimización de los caminos de acceso y el almacenamiento de los datos buscándose que el tiempo de respuesta sea el menor posible. Este tiempo de respuesta

está condic
puede habla
operadores
técnicas util
a los datos.

En la pró
una consulta
de acceso a
las mismas
permiten la
el operador
transformaci
optimizados
procesamien
consultas.

8.2 ESTABL

El paráme
involucra el
consulta y el

El costo
siguientes co

- *Costo d*
del o de
sitios en
un medi
la eficie
centraliz
- *Costo d*
tiempo
encuent

los diferentes pasos que
zar una consulta. Estos
encia, pues los mismos

onder a una consulta

ción de una consulta, se
tegias que van desde
a la optimización de los
s datos buscándose que
este tiempo de respuesta

está condicionado a dos hechos importantes. En primer lugar, se debe hablar de la forma en que se implementaron los diferentes operadores relacionales en el SABD. Y en segundo lugar, de las técnicas utilizadas que permiten la escogencia de los accesos óptimos a los datos.

En la próxima sección se establece el problema de la eficiencia de una consulta mediante un ejemplo en que se muestran dos estrategias de acceso a una misma consulta y se valoran los costos de acceso entre ellas. Posteriormente, se estudian algunos algoritmos que permiten la implementación de los operadores relacionales, sobre todo el operador join. Luego, se presenta un algoritmo que permite la transformación de los árboles que representan las consultas en árboles optimizados equivalentes. Finalmente, se introduce otra técnica de procesamiento que se denomina técnica de descomposición de consultas.

2 ESTABLECIMIENTO DEL PROBLEMA

El parámetro fundamental para medir la eficiencia de una consulta involucra el tiempo que transcurre entre el momento en que se hace la consulta y el momento en que se reciben los datos solicitados.

El costo total que debe minimizarse es entonces la suma de los siguientes costos [JARK84]:

- *Costo de comunicación.* Se refiere al costo de transmitir los datos del o de los sitios en donde se encuentran almacenados, al sitio o los sitios en donde se van a procesar. En un ambiente distribuido con un medio bajo de transmisión este puede ser un factor negativo en la eficiencia de la optimización. Por su parte, en un ambiente centralizado este factor es irrelevante.
- *Costo de acceso a la memoria secundaria.* Este costo se refiere al tiempo requerido para transmitir los datos -páginas- que se encuentran en la memoria secundaria, a los buffers del CPU.

- *Costo de almacenamiento.* Se refiere al costo del tiempo que transcurre para transmitir y confirmar las páginas de datos que se envían del CPU a la memoria secundaria.
- *Costo de procesamiento.* Es el costo referido al tiempo de ejecución o de utilización del CPU.

A continuación se introduce un ejemplo en que una consulta se representa mediante dos expresiones relacionales equivalentes y se evalúan los costos de cada una de ellas.

El costo se va a medir en función del número de tuplas intercambiadas entre los discos y la memoria principal. Se podrían considerar otros parámetros, como por ejemplo: la longitud de una tupla de una relación, el número de posibles valores de un atributo, la longitud de un atributo, etc.

Es importante mencionar que la unidad de transferencia entre los discos y el CPU es la página, que tiene un tamaño equivalente a un bloque físico. Sin embargo, para explicar el costo de procesamiento se considera solamente el intercambio de tuplas.

Ejemplo. Considerar la siguiente consulta:

Dar los nombres de los sitios visitados en viajes cuya ciudad de salida es Quito.

Dos posibles formas de expresar esta consulta son las siguientes:

Expresión 1: ((VIAJE * SITIO) : C-Salida = 'Quito') [Nom-Sitio]

Expresión 2: (((VIAJE : C-Salida = 'Quito') [No-Sitio]) * SITIO) [Nom-Sitio]

En la figura 8.2 se aprecian los árboles respectivos de dichas consultas.

Suponer que se tienen 200 tuplas en la relación SITIO y 10000 en la relación VIAJE, de las cuales 20 contienen a 'Quito' como valor en el atributo C-Salida.

Expresión 1:

*R1 = SITIO * VIAJE:* Este join requiere la lectura de 10000 tuplas de VIAJE

Además...
de ellas
interme...

R2 = R1
relación

Resultad...

Cantidad...

Expresió...

R1 = VI...
tuplas y s...

costo del tiempo que
páginas de datos que se

o al tiempo de ejecución

n que una consulta se
ales equivalentes y se

el número de tuplas
n principal. Se podrían
la longitud de una tupla
e un atributo, la longitud

transferencia entre los
año equivalente a un
sto de procesamiento se

ciudad de salida es Quito.

siguientes:

[Nom-Sitio]

o] * SITIO) [Nom-Sitio]

dichas consultas.

SITIO y 10000 en la relación
or en el atributo C-Salida.

de 10000 tuplas de VIAJE.

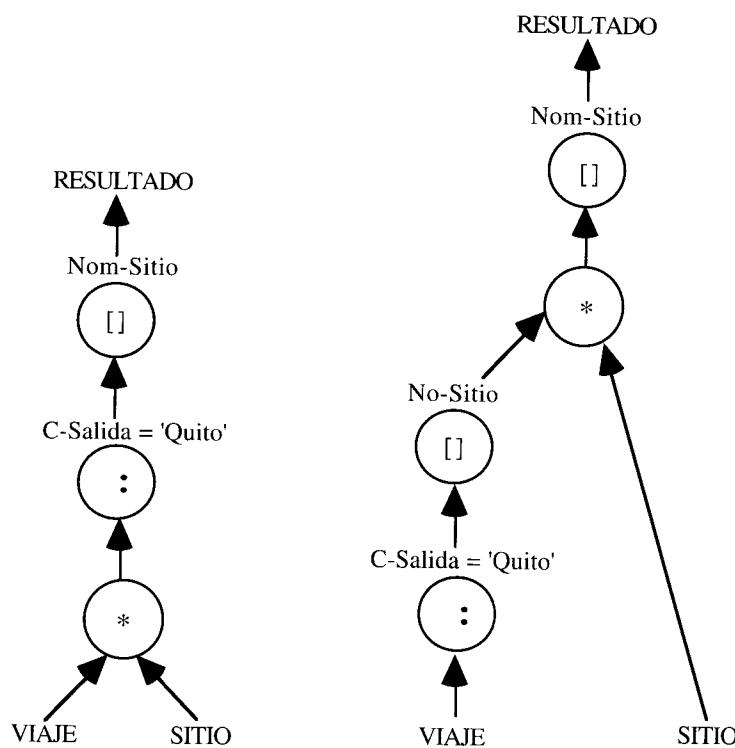


Figura 8.2 Ejemplo de dos árboles semánticos equivalentes

Además, se requiere la lectura de cada una de las 10000 tuplas de SITIO y cada una de ellas se debe hacer 10000 veces. De esta forma, se genera una relación intermedia de 2000000 tuplas y el envío de la misma a los discos.

$R2 = R1 : C\text{-Salida} = \text{'Quito'}$: Esta selección requiere la lectura de 200 tuplas de la relación intermedia R1. En este caso, se genera una relación de 20 tuplas.

Resultado = R2[Nombre-Sitio]: En este caso, se genera una relación de 20 tuplas.

Cantidad de tuplas involucradas : 2000200.

Expresión 2:

$R1 = VIAJE : C\text{-Salida} = \text{'Quito'}$: En este caso, se requiere la lectura de 10000 tuplas y se genera una relación de 20 tuplas.

$R2 = R1[No-Sitio]$: Se genera una relación de 20 tuplas.

$R3 = R2 * SITIO$: Lectura de 200 tuplas de SITIO y de 20 de R2. Así, se genera una relación intermedia de 4000 tuplas.

$Resultado = R3[Nombre-Sitio]$: Generación de una relación de 20 tuplas.

Cantidad de tuplas involucradas: 20200.

Este ejemplo muestra grandes diferencias en el costo de responder una consulta; pues la diferencia entre las dos consultas es del orden de 1:100. Por esta razón, es importante estudiar las estrategias que se han seguido para la implementación de los diferentes operadores relacionales en un SABD, principalmente el operador join, así como las técnicas que permitan escoger la expresión relacional de acceso más eficiente.

8.3 IMPLEMENTACION DE LOS OPERADORES RELACIONALES

8.3.1 Operador proyección

La implementación del operador proyección es directa. En efecto, sea $R(X)$ una relación y Y un subconjunto de atributos de X . En el caso de que se tenga una llave primaria en la relación R , el resultado será una lista de tuplas proyectadas a Y de igual cardinalidad que las tuplas originales de la relación R . Si, por el contrario, no se tiene una llave primaria, se debe proceder a la eliminación de las tuplas duplicadas.

8.3.2 Operador selección

Con respecto al operador selección, se pueden seguir distintas estrategias. Varios algoritmos clásicos de búsqueda de un archivo pueden utilizarse para ejecutar una selección en una relación.

Entre las estrategias de implementación del operador selección, se puede hablar de las siguientes:

- B
- T
- d
- I
- at
- tu
- In
- cc
- atr
- co

8.3.3 O

Con
diferenc
ordenam
son com

Para e
dos tupl
son com
barren la
toma una

Por s
cualesqu
 $m * n$ tu
de tuplas

Las ir
En realid
por qué e
involucra
deben co
cerca de l

as.
20 de R2. Así, se genera una
lación de 20 tuplas.

n el costo de responder
nsultas es del orden de
s estrategias que se han
operadores relacionales
sí como las técnicas que
eso más eficiente.

es directa. En efecto, sea
ntos de X. En el caso de
R, el resultado será una
inalidad que las tuplas
o, no se tiene una llave
as tuplas duplicadas.

pueden seguir distintas
eda de un archivo pueden
ación.

l operador selección, se

- *Barrido:* Esta es la estrategia más simple y consiste en estudiar cada tupla verificando la condición de selección; si la tupla la satisface se deja en la relación resultado, en caso contrario se desecha.
- *Indice:* En el caso de que la condición de selección involucre un atributo que es llave primaria, se usa el índice para llegar hasta las tuplas deseadas.
- *Indice secundario:* En el caso en que se tenga una igualdad en la condición de selección, se usa para recuperar una tupla si el atributo de selección es una llave primaria y para recuperar conjuntos de tuplas en caso contrario.

8.3.3 Operadores conjuntistas

Con respecto a los operadores conjuntistas -unión, intersección, diferencia, producto cartesiano- lo que se hace en términos generales es un ordenamiento en los atributos de las dos relaciones involucradas, los cuales son compatibles. Posteriormente se efectúa un barrido para dar el resultado.

Para el caso de la unión se comparan las relaciones y cuando aparecen dos tuplas iguales se deja una de las dos con el resto de las tuplas que no son comunes en las dos relaciones. Para el caso de la intersección, se barren las dos relaciones y cuando se encuentran las mismas tuplas, se toma una para dejarla en la relación resultado.

Por su parte, el producto cartesiano de dos relaciones R y S cualesquiera es sumamente costoso, pues el resultado del mismo incluye $m * n$ tuplas en donde m es el número de tuplas de R y n es el número de tuplas de S.

Las implementaciones de los operadores conjuntistas son costosas. En realidad cualquier operador binario es costoso. Por esta razón se verá por qué en cualquier técnica de optimización, los operadores binarios involucrados en el árbol de la expresión relacional asociada a la consulta deben colocarse lo más cerca posible de la raíz y los operadores unarios cerca de las hojas.

8.4 IMPLEMENTACION DEL OPERADOR Θ -JOIN

Es importante analizar la implementación de este operador en forma más profunda, ya que:

- Es el operador más importante del álgebra relacional
- Consumo bastante tiempo a la hora del procesamiento
- Como resultado del punto anterior es muy costoso.

De ahí la atención que se ha dado a la búsqueda de alternativas para su implementación.

En los diferentes algoritmos que se estudiarán a continuación, se utiliza la siguiente notación:

Sean $R(A, A_2, \dots, A_n)$ y $S(B, B_2, \dots, B_m)$ dos relaciones en donde los atributos A y B son compatibles. Además, sean t_1 y t_2 dos tuplas de R y S respectivamente y sea $RESULTADO = R \langle A \Theta B \rangle S$.

8.4.1 Ciclos iterados

El primero y más sencillo de los algoritmos que se utilizan para implementar el operador join es el llamado de los *ciclos iterados*. Con este algoritmo, cada tupla de una relación se compara con todas las tuplas de la otra relación verificándose si las tuplas coinciden en el atributo que sirve de join.

A continuación se presenta dicho algoritmo[MISH92]:

<i>Algoritmo de los ciclos iterados</i>
Entrada: Dos relaciones R y S
Salida: $R \langle A \Theta B \rangle S$
Para cada tupla t_2 en R hacer
{ Para cada tupla t_1 en S hacer
{ Si $t_1[A] \Theta t_2[B]$ entonces
Poner en la relación $RESULTADO$
la concatenación de t_1 y t_2 } }

OIN
te operador en forma

ncional
amiento
oso.

a de alternativas para

a continuación, se

ciones en donde los
t₂ dos tuplas de R y
S.

que se utilizan para
ciclos iterados. Con
mpara con todas las
las coinciden en el

H92]:

OO

La relación RESULTADO representa la relación de salida. El algoritmo de los ciclos iterados es sumamente costoso cuando las relaciones son grandes. Su costo es aproximadamente el costo de leer cada tupla de una relación multiplicado por el costo de leer todas las tuplas de la otra relación. En el caso de este algoritmo, la relación con mayor número de tuplas aparece en la parte interna.

Este algoritmo se ha implementado, entre otros, en los siguientes SABDs: Oracle 7 de Oracle, SQL Server de Sybase, DB₂ de IBM y Non-Stop SQL/MP de Tandem Computers.

En [MISH92] se destaca que si en dicho algoritmo, los atributos involucrados en el Θ-join pueden accesarse por medio de un índice, se puede mejorar considerablemente su rendimiento.

Ejemplo. En la figura 8.3, se muestra cómo se aplica el algoritmo de los ciclos iterados a dos relaciones. Se escoge la primera tupla de la relación GUIA -relación externa- [Juan Mora, Santiago] y se inicia el barrido de la relación SITIO -relación interna- hasta llegar a la primera tupla en que la concatenación es factible, es decir, la tupla [Desierto, Santiago]. La tupla producto de esa concatenación [Juan Mora, Santiago, Desierto, Santiago], se coloca en la relación RESULTADO. El proceso se sigue hasta barrer completamente la relación interna. Luego se toma la segunda tupla de la relación GUIA y el proceso se repite hasta que se haya barrido completamente esta relación.

GUIA		SITIO	
Nombre	Ciudad	Tipo	Ciudad
Juan Mora	Santiago	Desierto	Santiago
Juan Mora	Panamá	Mar/Litoral	Santiago
María Coto	Bogotá	Mar/Litoral	Caracas
Ana Salas	Santiago	Volcán	Bogotá

RESULTADO			
Nombre	Ciudad	Tipo	Ciudad
Juan Mora	Santiago	Desierto	Santiago
Juan Mora	Santiago	Mar/Litoral	Santiago
María Coto	Bogotá	Volcán	Bogotá
Ana Salas	Santiago	Desierto	Santiago
Ana Salas	Santiago	Mar/Litoral	Santiago

Figura 8.3 Aplicación del algoritmo de los ciclos iterados

8.4.2 Ordenamiento y fusión

Otra técnica utilizada para implementar el Θ -join es el algoritmo de *ordenamiento y fusión*, llamado así porque se compone precisamente de dos fases.

Cuando se utiliza este algoritmo, las relaciones se ordenan según los atributos involucrados en el Θ -join. Así, las dos relaciones se barren en dicho orden y las tuplas que satisfacen la condición se fusionan para generar una nueva relación.

A continuación, se presenta dicho algoritmo [MISH92]:

<i>Algoritmo de ordenamiento y fusión</i>
Entrada: Dos relaciones R y S
Salida: $R \langle A \Theta B \rangle S$
<i>Fase de ordenamiento</i>
Ordenar R sobre $t_1[A]$
Ordenar S sobre $t_2[B]$
<i>Fase de fusión</i>
Leer primera tupla de R;
Leer primera tupla de S;
Para cada tupla t_1 de R hacer
{ Mientras $t_2[B] < t_1[A]$ hacer
{ Leer siguiente tupla de S
{ Si $t_1[A] = t_2[B]$ entonces
Poner en la relación RESULTADO la
concatenación de t_1 y $t_2\}$ }}

Este algoritmo es superior al anterior, siempre y cuando las relaciones se ordenen previamente, por el sencillo hecho de que cada relación es barrida una sola vez. Así, el costo de este algoritmo es la suma de los costos de leer cada una de las relaciones.

Entre los SABDs que soportan este algoritmo se pueden citar: Oracle 7, Sybase SQL Server, DB₂ y Non-Stop SQL/MP [BONTE96].

Ejemplo. En la figura 8.4, se muestra cómo se aplica el algoritmo de ordenamiento y fusión a las relaciones del ejemplo anterior.

GUIA

Nombre	Ciudad
Juan Mora	Santiago
Juan Mora	Panamá
María Coto	Bogotá
Ana Salas	Santiago

SITIO

TIPO	Ciudad
Desierto	Santiago
Mar/Litoral	Santiago
Mar/Litoral	Caracas
Volcán	Bogotá

Fase de ordenamiento

GUIA

Nombre	Ciudad
María Coto	Bogotá
Juan Mora	Panamá
Juan Mora	Santiago
Ana Salas	Santiago

SITIO

TIPO	Ciudad
Volcán	Bogotá
Mar/Litoral	Caracas
Desierto	Santiago
Mar/Litoral	Santiago

Fase de fusión

RESULTADO

Nombre	Ciudad	Tipo	Ciudad
María Coto	Bogotá	Volcán	Bogotá
Juan Mora	Santiago	Desierto	Santiago
Juan Mora	Santiago	Mar/Litoral	Santiago
Ana Salas	Santiago	Desierto	Santiago
Ana salas	Santiago	Mar/Litoral	Santiago

Figura 8.4 Aplicación del algoritmo de ordenamiento y fusión

El algoritmo de ordenamiento y fusión se considera la mejor opción cuando se deben juntar relaciones muy grandes y donde el resultado también es una relación grande.

8.4.3 Hash

A continuación se introduce el llamado *algoritmo de hash* [MISH92]:

<i>Algoritmo de hash</i>
<p>Entrada: Dos relaciones R y S Salida: $R \langle A \ominus B \rangle S$</p>
<p>Para cada tupla t_2 en S hacer { Haga un hash en los atributos $t_2[B]$, Ponga la tupla en la tabla de hash dependiendo del valor producido} Para cada tupla t_1 hacer { Haga un hash en los atributos $t_1[A]$, Si $\text{hash}(t_1)$ cae en un lugar no vacío de la tabla entonces { Si $t_1[A] = t_2[B]$ entonces Poner en la relación RESULTADO la concatenación de t_1 y $t_2\}$ }</p>

En este algoritmo, a los valores de los atributos involucrados en el join de la primera relación se les aplica una función de hash, explotando así la rapidez con la que se pueden calcular las direcciones donde se encuentran las tuplas. Estos valores apuntan a una tabla hash en la cual cada entrada puede contener ya sean tuplas completas o solo identificadores de tuplas.

Entre los productos que soportan este algoritmo se pueden citar Oracle 7 versión 7.3 y NonStop SQL/MP.

8.5 OPTIMIZADOR DE CONSULTAS

Un *optimizador de consultas* es un conjunto de programas que tienen como finalidad la transformación de una expresión algebraica que representa una consulta de un usuario, en una expresión equivalente que sea menos costosa que la primera.

Así, se
expresio
permite
utilizand

8.5.1 Reg

A cont
expresione

r₁: Expl

Esta p
en donde
conjunció
equivalen
sub-condi

R :

En la fi

do algoritmo de hash

$t_2[B]$,
ash
ido}

$t_1[A]$,
vacío de

ULTADO
 $t_2\}$

s involucrados en el join de
hash, explotando así la
iones donde se encuentran
sh en la cual cada entrada
identificadores de tuplas.

goritmo se pueden citar

o de programas que tienen
expresión algebraica que
expresión equivalente que

Así, se van a establecer una serie de reglas de equivalencia de expresiones algebraicas para posteriormente presentar un algoritmo que permite transformar una expresión algebraica en una optimizada, utilizando para ello estas reglas de equivalencia.

8.5.1 Reglas de equivalencia

A continuación se van a introducir 10 reglas que se verifican en las expresiones algebraicas. Este conjunto de equivalencias no es exhaustivo.

r_1 : Explosión en cadena de la selección

Esta primera regla garantiza que una selección sobre una relación R, en donde la condición de selección E se puede expresar como una conjunción de sub-condiciones $E_1 \text{ AND } E_2 \text{ AND } \dots \text{ AND } E_n$, es equivalente a una aplicación reiterada de selecciones basadas en estas sub-condiciones, es decir, la siguiente equivalencia se cumple:

$$R : (E_1 \text{ AND } E_2 \text{ AND } \dots \text{ AND } E_n) \equiv (((R : E_1) : E_2) \dots) : E_n$$

En la figura 8.5 se representa gráficamente la equivalencia de esta regla.

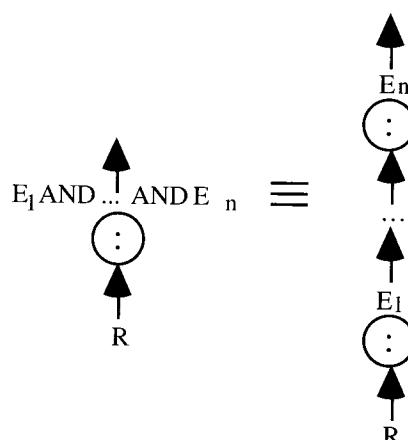


Figura 8.5 Representación gráfica de la regla r_1
(explosión en cadena de la selección)

r₂: Comutatividad de la selección

Como un corolario de la regla anterior, la siguiente equivalencia se verifica:

$$((R : E_1) : E_2) \equiv ((R : E_2) : E_1)$$

En la figura 8.6, se muestra la representación gráfica de esta regla.

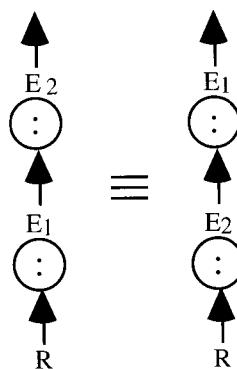


Figura 8.6 Representación gráfica de la regla r₂
(comutatividad de la selección)

r₃: Explosión en cadena de la proyección

Efectuar una serie de proyecciones sobre una misma relación es equivalente a realizar la proyección únicamente sobre el último conjunto de atributos proyectados, es decir, la siguiente equivalencia se cumple:

$$(((R[X_1])[X_2]) \dots [X_n]) \equiv R[X_n]$$

En este caso, para que las proyecciones reiteradas se cumplan se debe tener el siguiente orden de inclusiones: $X_1 \supset X_2 \supset \dots \supset X_n$.

En la figura 8.7, se muestra la representación gráfica de esta regla.

r₄: Transposición

Cuando se proyección, si encuentran en e proyección, la e

En la figura 8

guiente equivalencia se

E_1)

gráfica de esta regla.

regla r_2

una misma relación es
mente sobre el último
guiente equivalencia se

$X_n]$

adas se cumplan se debe
 $\supset \dots \supset X_n$.

gráfica de esta regla.

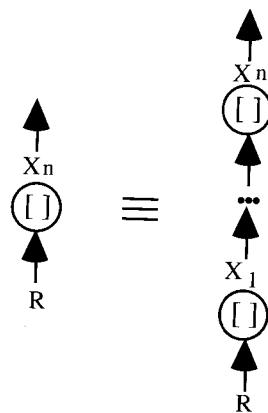


Figura 8.7 Representación gráfica de la regla r_3
(explosión en cadena de la proyección)

r_4 : Transposición de la proyección con la selección

Cuando se desea intercambiar el orden de una selección con una proyección, si los atributos ubicados en la condición de selección E se encuentran en el conjunto de atributos X sobre el cual se quiere hacer la proyección, la equivalencia que se verifica es la siguiente:

$$R[X] : E \equiv (R : E)[X]$$

En la figura 8.8, se muestra la representación gráfica de esta regla.

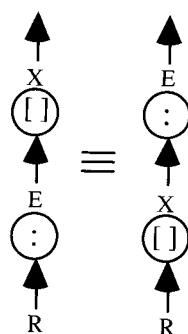
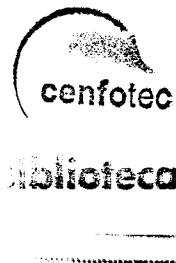


Figura 8.8 Representación gráfica de la regla r_4
(transposición de la proyección con la selección)



cenfotec

biblioteca

r₅: Transposición de la selección con el producto cartesiano (o con el Θ-join)

Cuando se desean invertir los operadores de selección y producto cartesiano en una expresión en donde se encuentran involucradas dos relaciones, según los atributos que aparecen en la condición de selección, se pueden caracterizar tres casos.

En primer lugar, si la condición E solo contiene atributos de una de las relaciones -suponer que sean los atributos de S-, entonces la equivalencia se expresa como:

$$(R \times S) : E \equiv R \times (S : E)$$

Por otra parte, si la condición E se puede representar como $E_1 \text{ AND } E_2$, en donde E_1 es una sub-condición que solo contiene atributos de R y E_2 solo atributos de S, entonces la equivalencia es la siguiente:

$$(R \times S) : E_1 \text{ AND } E_2 \equiv (R : E_1) \times (S : E_2)$$

Finalmente, si la condición E se puede representar como $E_1 \text{ AND } E_2$, en donde E_1 es una sub-condición que solo contiene atributos de R y E_2 contiene atributos de R y de S, entonces la equivalencia es la siguiente:

$$(R \times S) : E_1 \text{ AND } E_2 \equiv ((R : E_1) \times S) : E_2$$

Estas equivalencias también se cumplen si en vez del producto cartesiano se tiene el operador Θ-join debido a la equivalencia siguiente:

$$R \langle A \Theta B \rangle S = (R \times S) : (A \Theta B)$$

Entonces, según el caso, se tienen las siguientes equivalencias del operador Θ-join:

$$(R \langle A \Theta B \rangle S) : E \equiv R \langle A \Theta B \rangle (S : E)$$

$$(R \langle A \Theta B \rangle S) : E_1 \text{ AND } E_2 \equiv (R : E_1) \langle A \Theta B \rangle (S : E_2)$$

$$(R \langle A \Theta B \rangle S) : E_1 \text{ AND } E_2 \equiv ((R : E_1) \langle A \Theta B \rangle S) : E_2$$

Es
en
sele
E₂. E

Es
en Θ
r₆: T
ca

En
subco
atribu

Est
cartesi

En

producto cartesiano

de selección y producto
se encuentran involucradas dos
la condición de selección,

tiene atributos de una de
atributos de S-, entonces la

E)
representar como $E_1 \text{ AND } E_2$
o contiene atributos de R y
la es la siguiente:

$(R : E_1) \times (S : E_2)$

presentar como $E_1 \text{ AND } E_2$,
contiene atributos de R y E_2
equivalencia es la siguiente:

$(E_1 \times S) : E_2$

si en vez del producto
a la equivalencia siguiente:

$(A \Theta B)$

siguientes equivalencias del

$: E)$

$E_1 \langle A \Theta B \rangle (S : E_2)$

$E_1 \langle A \Theta B \rangle S : E_2$

En la figura 8.9 se representa el caso de la transposición de la selección y el producto cartesiano, en donde la condición $E = E_1 \text{ AND } E_2$, E_1 solo contiene atributos de R y E_2 solo atributos de S.

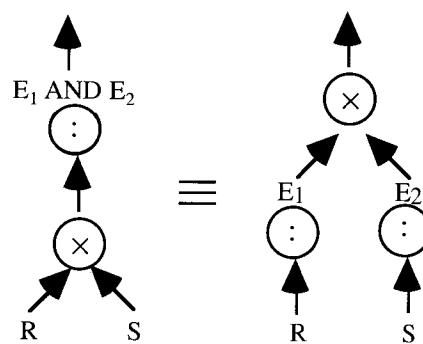


Figura 8.9 Representación gráfica de la regla r_5
(transposición de la selección con el producto cartesiano)

Es importante mencionar que las propiedades en donde se involucra un Θ -join también son aplicables al join.

r6: Transposición de la proyección con el producto cartesiano (o con el Θ -join)

En este caso, si R y S son dos relaciones y $X = X_1 \cup X_2$ es un subconjunto de atributos, en donde X_1 son atributos de R y X_2 son atributos de S, entonces se verifica la siguiente equivalencia:

$$(R \times S)[X] \equiv R[X_1] \times S[X_2]$$

Esta misma regla se cumple cuando se sustituye el producto cartesiano por el Θ -join, es decir, se tiene la siguiente equivalencia:

$$(R \langle A \Theta B \rangle S)[X] \equiv R[X_1] \langle A \Theta B \rangle S[X_2]$$

En la figura 8.10 se muestra la representación gráfica de esta regla.

r₈: A

El

cumpl

La

figura

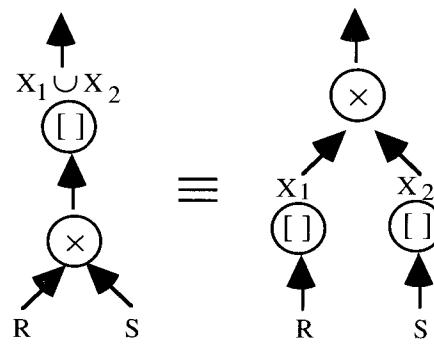


Figura 8.10 Representación gráfica de la regla r₆
(transposición de la proyección con el producto cartesiano)

r₇: Comutatividad de la unión y de la intersección

La comutatividad de los operadores de unión y de intersección son propiedades que se establecen por la definición misma de dichos operadores conjuntistas. Así, se tienen las siguientes equivalencias:

$$\begin{aligned} R \cup S &\equiv S \cup R \\ R \cap S &\equiv S \cap R \end{aligned}$$

La representación gráfica de la comutatividad de la unión y de la intersección se muestran en la figura 8.11.

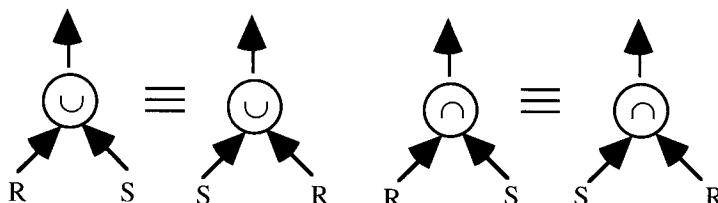


Figura 8.11 Representación gráfica de la regla r₇
(comutatividad de la unión y de la intersección)

r₈: Tra

com

Es lo

hacer p

posteri

se cumple

cumple

r₈: Asociatividad del Θ-join

El operador Θ-join es asociativo, es decir, la siguiente equivalencia se cumple:

$$(R \langle A \Theta B \rangle S) \langle C \Theta D \rangle T \equiv R \langle A \Theta B \rangle (S \langle C \Theta D \rangle T)$$

La representación gráfica de la asociatividad del join se muestra en la figura 8.12.

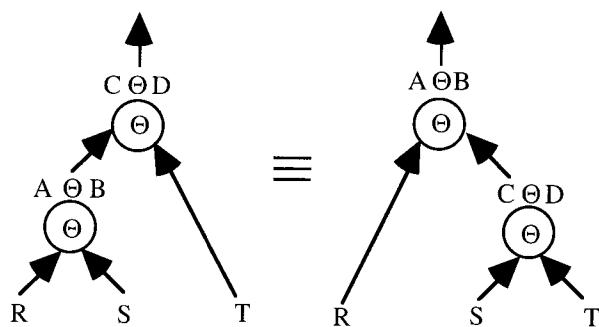


Figura 8.12 Representación gráfica de la regla r₈
(asociatividad del Q-join)

r₉: Transposición de la selección con la unión (con la intersección o con la diferencia)

Es lo mismo hacer una selección sobre una unión de relaciones, que hacer primero la selección sobre cada una de las relaciones y posteriormente efectuar la unión de estas relaciones. Esta regla también se cumple para los operadores intersección y diferencia, es decir, se cumple lo siguiente:

$$(R \cup S) : E \equiv (R : E) \cup (S : E)$$

$$(R \cap S) : E \equiv (R : E) \cap (S : E)$$

$$(R - S) : E \equiv (R : E) - (S : E)$$

En la figura 8.13 se muestra esta regla de equivalencia para el caso de la unión.

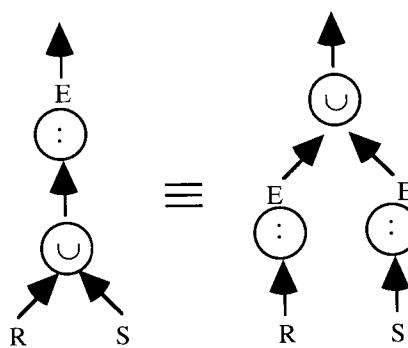


Figura 8.13 Representación gráfica de la regla r_9
(transposición de la selección con la unión)

r_{10} : Transposición de la proyección con la unión

En la regla de transposición de la proyección con respecto a la unión, su equivalencia se presenta así:

$$(R \cup S)[Z] \equiv R[Z] \cup S[Z]$$

En la figura 8.14 se muestran estas dos reglas de equivalencia.

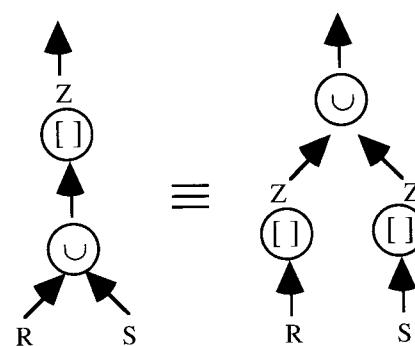


Figura 8.14 Representación gráfica de la regla r_{10}
(transposición de la proyección con la unión)

alencia para el caso de

E
S

egla r_9
ión)

ón
con respecto a la unión,

Z]

de equivalencia.

Z
[]
S

regla r_{10}
unión)

Esta regla también es aplicable a la intersección y a la diferencia, es decir, las siguientes equivalencias se verifican:

$$(R \cap S)[Z] \equiv R[Z] \cap S[Z]$$

$$(R - S)[Z] \equiv R[Z] - S[Z]$$

A continuación, se va a presentar, en forma general, el algoritmo de optimización introducido por Smith y Chang [SMIT75] que permite transformar una expresión algebraica en una equivalente optimizada.

8.5.2 Algoritmo de optimización de expresiones algebraicas

La estrategia de base de dicho algoritmo es trasladar los operadores más costosos cerca de la raíz, es decir, del resultado. Además, ubicar los operadores menos costosos -proyección y selección- lo más cerca posible de las hojas, es decir, de las relaciones de base y vistas involucradas en la expresión.

Los operadores unarios no solo son menos costosos sino también son de *carácter reductivo*, es decir, reducen el tamaño de las relaciones participantes antes de aplicar los operadores binarios.

Este algoritmo se establece sobre 5 pasos, los cuales se enuncian a continuación:

Paso 1:

Separar las selecciones que contienen varios predicados en forma conjuntiva, usando la regla r_1 .

Paso 2:

Descender las selecciones lo más bajo posible en el árbol, con ayuda de las reglas r_4 , r_5 y r_9 .

Paso 3:

Agrupar las selecciones que se realizan sobre una misma relación usando la regla r_1 .

Paso 4:

Descender las proyecciones lo más bajo posible en el árbol, con ayuda de las reglas r_6 y r_{10} .

Paso 5:

Agrupar las proyecciones sucesivas dejando los atributos restantes y eliminar eventuales proyecciones inútiles que hubiesen podido aparecer (proyección sobre todos los atributos de una relación usando la regla r_3).

Ahora, por medio del siguiente ejemplo se va estudiar cómo dicho algoritmo se puede aplicar a una consulta de usuario.

Ejemplo. Sea la base de datos sobre el Club de Ecoturismo y suponer que se desea responder a la siguiente consulta:

Dar la lista de los nombres de turistas que visitaron un sitio en el continente africano después del 1 de mayo de 1996.

Una respuesta a esta consulta es la siguiente:

$((\text{TURISTA} * \text{VIAJE}) * \text{SITIO}) : \text{Continente} = \text{'Africa'}) : E[\text{Nombre-Turista}]$

con $E = \text{'Fecha-Salida'} > 05/01/96$

El árbol correspondiente a esta expresión se aprecia en la figura 8.15.

Siguiendo los diferentes pasos del algoritmo aplicados a este ejemplo, se obtiene el árbol optimizado que aparece en la figura 8.16.

8.6 DESCOMPOSICION DE CONSULTAS

En el sistema INGRES, el procesamiento de una consulta, el cual se explica en [STON76] y se resume en esta sección, se hace a partir de un proceso elegante que ejecuta dos funciones.

En primer lugar, se tiene la *descomposición* cuyo objetivo es descomponer una consulta que involucra distintas variables, en una

sucesión de s
Esta función

Luego, se
una sola varia
One Variable

8.6.1 Programación

El program

- la sustituci
- la separaci
- el reforma

e en el árbol, con ayuda

os atributos restantes y que hubiesen podido de una relación usando

va estudiar cómo dicho cuadro.

ismo y suponer que se desea

on un sitio en el continente

a') : E)[Nombre-Turista]

la figura 8.15.

a este ejemplo, se obtiene el

una consulta, el cual se
n, se hace a partir de un

ción cuyo objetivo es
tintas variables, en una

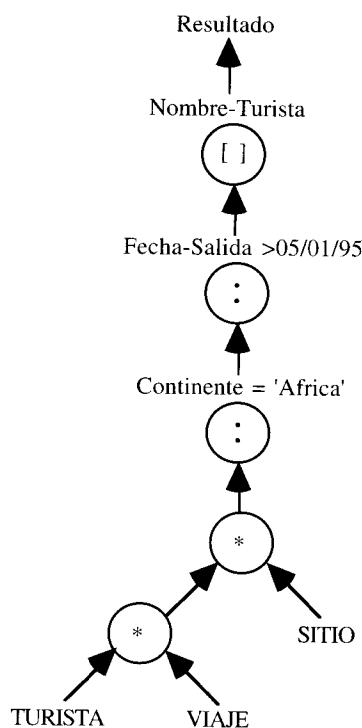


Figura 8.15 Árbol semántico de la consulta

sucesión de sub-consultas que contienen cada una, una sola variable. Esta función la realiza un programa denominado DECOMP.

Luego, se tiene el *procesamiento* de cada sub-consulta que involucra una sola variable por medio de un programa llamado *OVQP* -del inglés *One Variable Query Processor*.

8.6.1 Programa DECOMP

El programa DECOMP se sustenta sobre tres técnicas:

- la sustitución de una tupla
- la separación de una variable
- el reformato

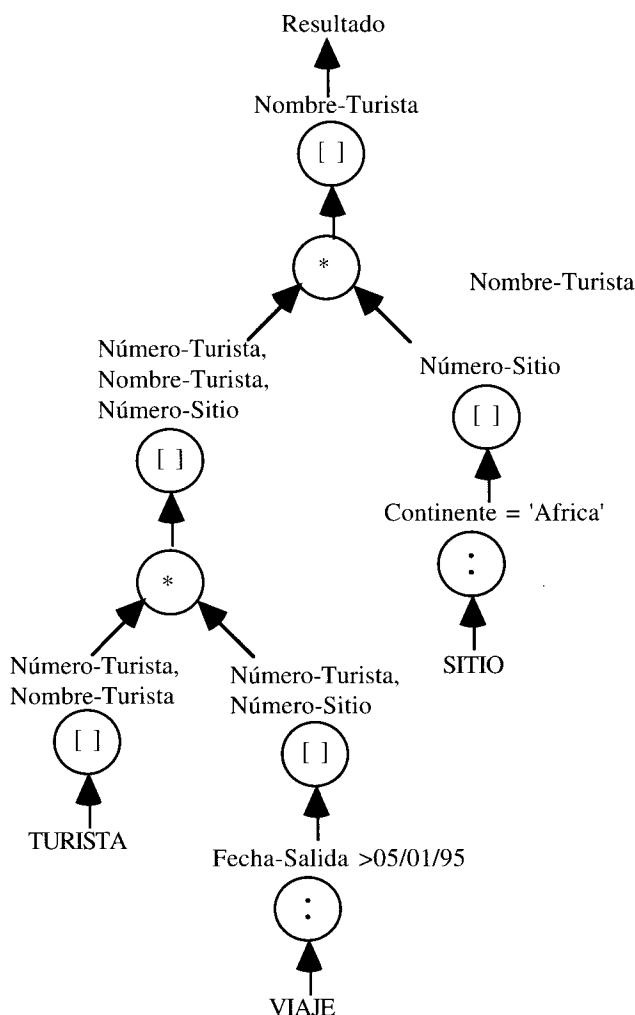


Figura 8.16 Árbol optimizado del la figura 8.15

La sustitución de una tupla

El objetivo es reducir una consulta por otras que tengan menos variables. Así, una variable en la consulta se selecciona para ser sustituida. El lenguaje AMI -del inglés Access Methods Interface- se utiliza para hacer un barrido

de las tuplas, la consulta que solo

La separación

Si la consulta es de la forma

se puede:

1. reemplazar el WHERE

En la consulta se recogen los procesos

2. Recuperar el resultado

La variable Var

El reformulamiento

Si la tupla para una consulta puede refinar esta forma

8.6.2 Programación

Por su parte las tuplas de

de las tuplas, una a la vez, de la relación asociada a la variable. Para cada tupla, los valores de los atributos de esta relación son sustituidos en la consulta, la cual cuenta con una variable menos. Este proceso se sigue hasta que solo quede una variable, en cuyo caso se llama al programa OVQP.

La separación de una variable

Si la condición C que acompaña el **where** en una consulta en QUEL, es de la forma

$$C_1(V_1) \text{ AND } C_2(V_2, \dots, V_n)$$

se puede descomponer la variable de tupla V_1 , en dos sub-consultas:

1. **retrieve into D (L[V₁])**
where C₁(V₁)

En este caso **L** representa la lista de los atributos que son requeridos por el resto de la consulta, es decir, C_2 y se pasa al programa OVQP.

2. Reemplazar R_1 , la relación que se asocia a la variable V_1 por **D** en el **range** y suprimir $C_1[V_1]$ de la condición **C**.

La técnica de separación de una variable (*OVD*) -del inglés *One Variable Detachement*-, tiene como efecto reducir el tamaño de la relación que interviene en la consulta por proyección y selección.

El reformato

Si la técnica OVD es una operación factible después de la sustitución para una variable V_1 restante, entonces la relación R_1 asociada a V_1 , puede reformatearse para tener los atributos de $C_1(V_1)$ como llave. De esta forma se puede iniciar OVD cada vez que se realice una sustitución.

8.6.2 Programa OVQP

Por su parte, el programa OVQP permite un acceso eficiente a las tuplas de una relación dada sobre una consulta de una variable.

La parte inicial de este programa se denomina STRATEGY y sirve para determinar:

- qué llave debe ser eventualmente utilizada para accesar a la relación,
- qué valores de esta llave se usarán para llamar a la rutina FIND -que permite determinar un identificador de la tupla en una página de datos- de AMI,
- si el acceso debe hacerse directamente vía AMI sobre la estructura de almacenamiento de la relación de base o si se hace vía un índice secundario -en este caso STRATEGY debe escoger uno de los posibles índices-.

Así, las tuplas se recuperan mediante la estrategia de acceso que se seleccionó y se procesan por medio de una parte del programa OVQP denominado SCAN.

8.6.3 Algoritmo de descomposición

A continuación se presenta el algoritmo de descomposición utilizado en INGRES y posteriormente se introduce un ejemplo con el fin de comprender su funcionamiento.

<i>Algoritmo de descomposición</i>	
<i>Paso 1</i>	
	Si el número de variables en la condición es 0 o 1, entonces llamar a OVQP y termine; en caso contrario siga con el paso siguiente.
<i>Paso 2</i>	
	Encontrar las variables V_1, V_2, \dots, V_n que correspondan a consultas con una sola variable para aplicar OVD. La nueva relación para cada variable V_i se almacena como un archivo hash con una llave que se escoge de la siguiente forma:
2.1	Para cada j , seleccionar las condiciones C_{ij} de la forma $V_i.d_i = V_j.d_j$ en donde d_i y d_j son dominios en V_i y V_j .
2.2	Formar la llave K_i de la concatenación de los dominios d_{i1}, d_{i2}, \dots de V_i que aparecen en la condición C_{ij} .
2.3.	Si existe más de un j , se escoge un C_{ij} arbitrario no vacío para formar la llave. Si C_{ij} es vacío, para todo j la relación se almacena sin ningún orden.

Cuadro continua

Paso 3
Escoger
Paso 4
Reformular
diente se
Paso 5
Para cada
• cam
• llan
la ce
• fusio

En el
descomp

Ejemp

Dar la
del mis
Montrea

range o
range or
range of
retrieve
where s
and s.co
and v.c
and s1.n
and s.nu

El grafo c

Paso 1. L

Paso 2. S
consultas
consulta 1

range
retrieve
where

a STRATEGY y sirve

a accesar a la relación,
a la rutina FIND -que
upla en una página de

AMI sobre la estructura
si se hace vía un índice
de escoger uno de los

ategia de acceso que se
te del programa OVQP

escomposición utilizado
ejemplo con el fin de

nces llamar a OVQP y

a consultas con una sola
a variable V_1 se almacena
a siguiente forma:

rrma $V_i \cdot d_i = V_j \cdot d_j$ en donde

nios d_{i1}, d_{i2}, \dots de V_i que

o vacío para formar la
na sin ningún orden.

• Cuadro continúa...

Paso 3

Escoger la variable con el menor número de tuplas para realizar la técnica de sustitución.

Paso 4

Reformatear si es necesario las otras relaciones de forma que la llave correspondiente sea la concatenación de los atributos que aparecen en los predicados asociados.

Paso 5

Para cada tupla de la variable seleccionada en el paso 3

- cambiar los valores de la tupla en la consulta,
- llamar el algoritmo de descomposición en forma recursiva sobre una copia de la consulta resultante la cual se ha reducido a una variable,
- fusionar los resultados del punto anterior con las iteraciones anteriores.

En el siguiente ejemplo se muestra cómo se aplica el algoritmo de descomposición.

Ejemplo. Considerar la siguiente consulta y su respuesta en el lenguaje QUEL

Dar la lista de los nombres y los tipos de los sitios ubicados en Africa que son del mismo tipo que el sitio 345 y que pueden visitarse a partir de la ciudad de Montreal.

```
range of s is sitio
range of s1 is sitio
range of v is viaje
retrieve (s.nom_sitio, s.tipo)
where s. tipo = s1.tipo
and s.continente = 'Africa'
and v.c_salida = 'Montreal'
and s1.num_sitio = 345
and s.num_sitio = v.num_sitio
```

El grafo de esta consulta se puede representar por medio de la figura 8.17.

Paso 1. La consulta no es de una sola variable.

Paso 2. Se determinan las variables -en este caso S, S₁, V- para establecer subconsultas de una sola variable y así aplicar la técnica OVD.

consulta 1:

```
range of s is sitio
retrieve into d1(s.num_sitio, s.nom_sitio, s.tipo)
where s. continente = 'Africa'
```

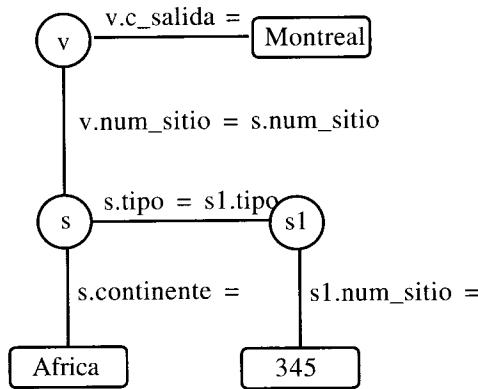


Figura 8.17 Grafo de la consulta ejemplo

consulta 2:

```

range of s1 is sitio
retrieve into d2(s1.num_sitio, s1.tipo)
where s1. num_sitio = 345
  
```

consulta 3:

```

range of v is viaje
retrieve into d3(v.num_sitio)
where v. c_salida = 'Montreal'
  
```

Una vez que se separan estas sub-consultas, la consulta original se reduce a una cuya grafo aparece en la figura 8.18.

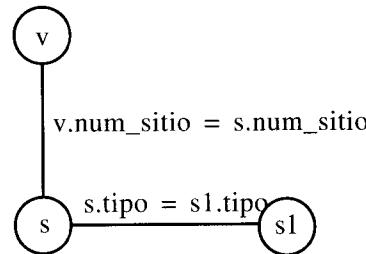


Figura 8.18 Grafo después de separar las consultas con selecciones constantes

Las relaciones d_1 , d_2 y d_3 asociadas a las variables S , S_1 y V respectivamente se almacenan bajo una organización hash. Para el caso de d_3 el sistema escoge como llave *Num_Sitio*.

Para el caso de d_1 y d_2 puede escogerse como llave *Tipo* ($s.\text{tipo} = s1.\text{tipo}$) o *Num_Sitio* ($s.\text{num_sitio} = v.\text{num_sitio}$). Suponer que se escoge *Tipo* como llave en la relación d_1 y *Num_Sitio* en d_2 .

La consulta inicial queda de la siguiente forma:

```
range of s is d1  
range of s1 is d2  
range of v is d3  
retrieve (s.nom_sitio, s.tipo)  
where v.num_sitio = s.num_sitio  
and s.tipo = s1.tipo
```

Paso 3. Escoger la variable con el número más pequeño de tuplas para realizar el método de sustitución de una tupla, con el fin de reducir el número de llamadas al OVQP. En este caso $s1$ tiene una sola tupla y se escoge para la sustitución.

Paso 4. Reformatear d_1 tomando como llave *Num_Sitio*, pues la escogencia de *Tipo* como llave no era la correcta.

Paso 5. Para cada tupla de la variable que se seleccionó en la etapa 3, se debe:

- Cambiar la variable que se separó por los valores de sus atributos en la consulta (en este caso valor por $s1.\text{tipo}$).

```
range of s is d1  
range of v is d3  
retrieve (s.nom_sitio, s.tipo)  
where v.num_sitio = s.num_sitio  
and s.tipo = (valor)
```

- Regresar al paso 1 llamando recursivamente el algoritmo de descomposición.
- Fusionar los resultados.

Siguiendo el proceso de descomposición en forma similar se dan los siguientes pasos:

Paso 1. La consulta no tiene una sola variable

Paso 2.

consulta 4:

```
range of e is d3
retrieve into d4(e.num_sitio, e.nom_sitio)
where tipo = (valor)
```

Así, la consulta queda

```
range of v is d3
range of e is d4
retrieve (e.nom_sitio)
where e.num_sitio = v.num_sitio
```

Paso 3. Suponer que D_4 tiene menos tuplas que D_3 . Se usa entonces para aplicar el método de sustitución.

Paso 4. El reformateo no es necesario

Paso 5. Se reemplaza Nom_Sitio, Num-Sitio por sus valores -valor 1- y -valor 2-. Se obtiene así:

consulta 5:

```
range of v is d3
retrieve (valor 1)
where (valor 2) = v.num_sitio
```

El grafo respectivo se muestra en la figura 8.19. En este caso la descomposición no se puede aplicar y se da por concluida la descomposición

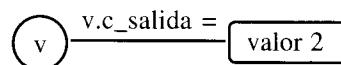


Figura 8.19 Grafo de una consulta irreducible

Resumiendo, la consulta

```
range of s is sitio
range of s1 is sitio
range of v is viaje
```

```
retrieve (s.nom_sitio, s.tipo)
where s.tipo = s1.tipo
and s.continente = 'Africa'
and v.c_salida = 'Montreal'
and s1.num_sitio = 345
and s.num_sitio = v.num_sitio
```

se descompone en las siguientes subconsultas:

```
range of s is sitio
retrieve into d1(s.num_sitio, s.nom_sitio, s.tipo)
where s.continente = 'Africa'
```

```
range of s1 is sitio
retrieve into d2(s1.num_sitio, s1.tipo)
where s1.num_sitio = 345
```

```
range of v is viaje
retrieve into d3(v.num_sitio)
where v.c_salida = 'Montreal'
```

```
range of e is d3
retrieve into d4(e.num_sitio, e.nom_sitio)
where tipo = (valor)
```

```
range of v is d3
retrieve (valor 1)
where (valor 2) = v.num_sitio
```

Se usa entonces para aplicar el

sus valores -valor 1- y -valor 2-.

En este caso la descomposición no
es correcta.

lor 2

ta irreducible

EJERCICIOS Y PREGUNTAS DE REPASO

- 8.1 Explicar los diferentes pasos involucrados en el procesamiento de una consulta.
- 8.2 Sean X y Y subconjuntos de atributos de R y S respectivamente. Demostrar las siguientes equivalencias de expresiones algebraicas:
 - a. $R(X, Y) \div S(Y) \equiv R[X] — ((R[X] \times S[Y]) — R(X, Y))[X]$

b. $(R \times S)[X, Y] \equiv R[X] \times S[Y]$

c. $(R \times S)[X] \equiv R[X] \times S$

d. $(R \times S)[Y] \equiv R \times S[Y]$

- 8.3 Sea $R(A_1, A_2, \dots, A_n)$ una relación definida sobre los dominios D_1, D_2, \dots, D_n . Se define el complemento de R al siguiente conjunto de tuplas $\neg R = \{t / t \in D_1 \times D_2 \times \dots \times D_n \text{ y } t \notin R\}$. Mostrar las siguientes equivalencias [DELO82]:

a. $\neg R(A_1, \dots, A_n) \equiv R[A_1] \times \dots \times R[A_n] — R(A_1, \dots, A_n)$

b. $R \cap S \equiv \neg (\neg (R) \cup \neg (S))$

- 8.4 Aplicar el algoritmo de optimización a la siguiente expresión algebraica:

$((TURISTA * SITIO) * VIAJE : (País = 'Honduras' \text{ AND } Fecha-Salida < 12/01/96 \text{ AND } Tipo = 'Volcán' \text{ AND } Continente = 'Asia'))[\text{Nombre-Turista}]$

que traduce la siguiente consulta:

Dar la lista de los turistas que son hondureños y que han visitado antes del 1 de diciembre de 1996, un volcán que se encuentra en el continente asiático.

- 8.5 Considerar que la relación TURISTA tiene 2000 tuplas, SITIO 1500 tuplas y VIAJE 3000 tuplas. Calcular el join de TURISTA *VIAJE*SITIO y aplicar los algoritmos de ordenamiento y fusión, de ciclos iterados y de hash y establecer cuál es más eficiente en este caso.

da sobre los dominios D_1, \dots, D_n .
R al siguiente conjunto de
R}. Mostrar las siguientes

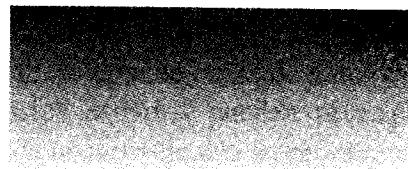
J — R(A₁, ..., A_n)

a la siguiente expresión

= ‘Honduras’ AND Fecha-
volcán’ AND Continente =

dureños y que han visitado
volcán que se encuentra en el

A tiene 2000 tuplas, SITIO
icular el join de TURISTA
os de ordenamiento y fusión,
cer cuál es más eficiente en



9

Control de la Concurrencia

«Para mí no hay duda de que el pensar se desarrolla en su mayor parte sin el uso de signos (palabras), y por encima de ello y en un grado considerable, de una forma inconsciente. ¿Pues cómo puede ocurrir, si no, que a veces “nos extrañemos” espontáneamente ante un suceso determinado? Este “extrañarse” parece surgir allí donde un determinado suceso entra en conflicto con un mundo conceptual suficientemente fijado entre nosotros. Cuando este conflicto es vivido dura e intensamente repercute de un modo decisivo sobre nuestro mundo de pensamientos. El desarrollo de este mundo de pensamientos es en cierto modo una huida continua del “extrañarse”.»

Albert Einstein (1879-1955),
físico estadounidense.

9.1 INTRODUCCION

Uno de los temas más importantes en un sistema de bases de datos es el referente a la posibilidad de que varios usuarios puedan accesar simultáneamente la base de datos. Así, cada usuario ejecuta su programa -denominado transacción- en forma transparente, es decir, como si fuera el único usuario que accesa la base de datos.

En un ambiente secuencial, las transacciones se realizan una a una, es decir, que una transacción se lleva a cabo en forma única y otras no pueden hacerlo hasta tanto la transacción en cuestión, no haya concluido. Es evidente que un ambiente de este tipo es a todas luces inconveniente -por su bajo rendimiento y altos costos- para los sistemas distribuidos actuales.

Así, es imperativo contar con la posibilidad de que el SABD permita que varias transacciones se realicen en forma concurrente. Sin embargo, la concurrencia puede conducir a varios problemas o conflictos. Los conflictos surgen cuando dos o más transacciones se interesan simultáneamente por un mismo objeto de la base de datos -relación, tupla, atributo, etc.-.

Cuando se realizan varias transacciones en forma concurrente se dice que se tiene un *itinerario*. Así, el SABD cuenta con varios mecanismos -protocolos, asignación de estampillas de tiempo, etc.- para coordinar dicha concurrencia y resolver eventuales conflictos.

En la próxima sección se introduce el concepto de transacción. Posteriormente se establecen los posibles conflictos que se presentan cuando dos transacciones o más se interesan simultáneamente por los mismos objetos. Además, se define el concepto de itinerario y se presentan las diferentes alternativas para establecer itinerarios consistentes, es decir, itinerarios que dejen la base de datos en un estado consistente. El capítulo termina con una introducción de las llamadas transacciones de larga duración y los problemas inherentes que se dan en tales casos.

9.2 TRANSACCION

Una *transacción* se define como la ejecución de un programa que accesa una base de datos la cual es compartida por varios usuarios en forma simultánea.

En el siguiente ejemplo se hace una introducción intuitiva de este concepto.

Ejemplo. Sean las siguientes relaciones:

PROFESOR(Num-Profesor, Nom-Profesor, Puesto, Salario, Nom-Depto)

DEPARTAMENTO(Num-Depto, Nom-Depto, Planilla)

Una transacción que incremente el salario de los profesores del Departamento de Biología Marina en un 10 % y que además haga que dichos incrementos se reflejen en el atributo Planilla, el cual contiene la nómina de cada Departamento, se puede presentar de la siguiente forma:

```
inicio de la transacción Incremento
  actualizar la relación profesor
  hacer salario = salario * 1.1
  donde nom_dept = Biología Marina
  actualizar la relación departamento
  hacer planilla = planilla * 1.1
  donde nom_dept = Biología Marina
fin de la transacción Incremento
```

Una transacción se puede definir como una sucesión finita de operaciones o_i

$$o_1 \rightarrow o_2 \rightarrow \dots \rightarrow o_n$$

que se realizan sobre un conjunto de objetos de la base de datos -relaciones, atributos, tuplas-.

Con respecto a las operaciones que se pueden efectuar sobre dichos objetos, entre otras, se tienen las siguientes:

- **begin transaction** inicio de la transacción

n de un programa que
por varios usuarios en
cción intuitiva de este

ario, Nom-Depto)

resores del Departamento de
hos incrementos se reflejan
da Departamento, se puede

na sucesión finita de

ase de datos -relaciones.

n efectuar sobre dichos

- **read a** lectura del objeto *a*
- **write a** escritura del objeto *a*
- **rollback** anulación de la transacción
- **commit** fin de una transacción.

Asimismo, una transacción se puede ver como una sucesión de operaciones en donde la primera es el inicio de la transacción -**begin**- y la última el fin de la misma -**commit**- . Las operaciones restantes es lo que se llamará el *cuerpo de la transacción* [MIRA86], según se aprecia en la figura 9.1.

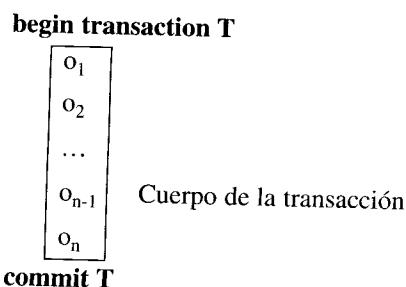


Figura 9.1 Cuerpo de una transacción

Una noción importante de este tema es el referente a la atomicidad de las transacciones. En efecto, una transacción representa una unidad atómica de ejecución, es decir, que dicha transacción, o se realiza completamente, o no se realiza del todo. Este concepto se profundizará en el capítulo siguiente cuando se estudien los diferentes mecanismos utilizados por un SABD para la recuperación de una base de datos y para dejarla en un estado consistente, en caso de que ocurra alguna falla.

El estado de una base de datos se dice que es *consistente* si satisface todas las consideraciones iniciales de estructura e integridad. Cuando se realiza una transacción, en principio, el estado de la base de datos cambia, pero deja a la misma en un estado consistente.

En la figura 9.2 se puede apreciar cómo una base de datos que se encuentra en un estado E_i en un tiempo t_i pasa a un estado E_j en un tiempo t_j después de haberse llevado a cabo la transacción T .

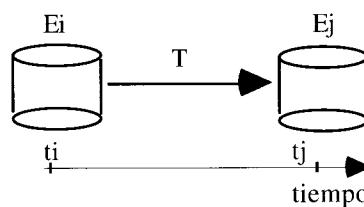


Figura 9.2 Estado consistente de una base de datos

Ejemplo. Sea una base de datos sobre las personas que son ahorrantes en un banco y conformada por la siguiente relación:

CLIENTE

Número	Nombre	Monto
2000	Juan Mora	40000
2456	María Mena	90000
4546	Carlos Salas	25000

Por otra parte suponer que se desea trasladar el 10 % del ahorro de Juan Mora y Carlos Salas a María Mena.

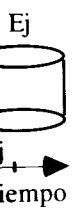
Para ello se realiza la siguiente transacción que se llamará *Transferencia*:

```

begin transaction Transferencia
  read a
  read b
  read c
  b = b + a*0.1 + c*0.1
  a = a - a*0.1
  c = c - c*0.1
  write a
  write b
  write c
commit Transferencia

```

una base de datos que se pasa a un estado E_j en una transacción T.



s que son ahorrantes en un banco

Monto
40000
90000
25000

10 % del ahorro de Juan Mora y
llamará Transferencia:

Al finalizar la transacción, el nuevo estado de la base de datos se refleja en la relación CLIENTE:

CLIENTE		
Número	Nombre	Monto
2000	Juan Mora	36000
2456	María Mena	96500
4546	Carlos Salas	22500

Se puede ver que dicha base de datos ha quedado en un estado consistente pues la suma de los montos de la relación es la misma que la relación anterior.

Resumiendo, se puede decir que una transacción tiene las siguientes cuatro propiedades, conocidas como *a.c.i.d.* [HAER83]:

- *atomicidad*: una transacción debe tener un papel de todo o nada en la base de datos, es decir, o los efectos de la transacción completa se dejan o se desechan todos estos efectos.
- *consistencia*: los efectos de una transacción trasladan la base de datos de un estado consistente a otro estado consistente.
- *independencia*: una transacción se realiza como si fuera la única en hacerlo.
- *durabilidad*: los efectos de una transacción que ha alcanzado su punto de validación -ha alcanzado el **commit**-, deben ser permanentes en la base de datos.

9.3 CONFLICTOS ENTRE TRANSACCIONES CONCURRENTES

Cuando se realiza una transacción, ésta puede interesarse en un objeto de dos formas diferentes: ya sea para leerlo o bien para escribir sobre este objeto.

Por otra parte, en el caso de que dos transacciones se interesen en forma simultánea por un objeto a , se tendrían cuatro posibilidades.

En primer lugar, puede darse el caso de lectura-lectura, en que las dos transacciones emiten una operación de lectura sobre el objeto a . En este caso, se obtendría el mismo valor y por lo tanto no existiría conflicto.

Sin embargo, deben estudiarse los tres casos restantes, es decir:

- escritura-escritura
- escritura-lectura
- lectura-escritura

pues éstos pueden provocar conflictos, como se verá en lo que sigue.

Los ejemplos de los siguientes apartados están inspirados en [DELO82] y [DATE86].

9.3.1 Escritura-escritura

Es cuando dos transacciones T_1 y T_2 hacen una escritura sobre el mismo objeto a . En este caso se puede perder la actualización hecha por alguna de las transacciones, como se aprecia en el siguiente ejemplo.

Ejemplo. En la figura 9.3, se esquematiza en una tabla de cuatro columnas, dos transacciones que se realizan concurrentemente.

En la tercera columna aparecen los valores reales que toman los objetos involucrados, es decir, los valores iniciales y los valores finales obtenidos cuando las dos transacciones han llegado a su punto de validación. Además, en la cuarta columna aparecen los valores esperados, es decir, los verdaderos valores que deberían tomar los objetos involucrados si se quiere que la base de datos se mantenga en un estado consistente.

Por otra parte, cada línea representa una acción; las acciones se encuentran ordenadas en el tiempo.

El ejemplo presenta dos transacciones que incrementan sucesivamente el monto de una cuenta de ahorros.

El problema que se puede ver en este caso es el hecho de que el valor actualizado por la transacción T_2 ($a = 5000$) se pierde ya que T_1 hace una escritura sobre a y el valor final es $a = 8000$ y no 10000 , que sería el valor esperado para mantener la base de datos en un estado consistente.

lectura-lectura, en que las dos transacciones realizan una lectura sobre el objeto a . En este caso no existiría conflicto. Los demás casos restantes, es decir:

no se verá en lo que sigue.

artados están inspirados en

hacen una escritura sobre el mismo objeto. Considerar la actualización hecha por otra transacción se ve en el siguiente ejemplo.

una tabla de cuatro columnas, dos de ellas para los valores reales que toman los objetos y las otras dos para los valores finales obtenidos cuando se ha llegado al punto de validación. Además, en la cuarta columna se indican los valores que se han escrito en la base de datos y en la quinta columna se indica el valor esperado para mantener la base consistente.

En este caso se presentan las acciones se encuentran en orden cronológico; las acciones se suceden sucesivamente el monto de la actualización.

El hecho de que el valor actualizado por la transacción T_1 hace una escritura sobre a y el valor esperado para mantener la base consistente.

tiempo	T_1	T_2	valor real	valor esperado
t1		begin T₂		
t2	begin T₁			
t3		read a		
t4	read a			
t5			$a = 3000$	
t6			$a = 3000$	
t7			$a = a + 2000$	
t8			$a = 5000$	
t9		write a		
t10		commit T₂		
			$a = 5000$	$a = 5000$
			$a = 8000$	$a = 10000$

Figura 9.3 Ejemplo de pérdida de actualización

De este ejemplo se deduce que una transacción no debe escribirse sobre un objeto en donde el valor ha sido modificado por otra transacción sin que ésta no haya llegado a su punto de validación.

9.3.2 Escritura-lectura

Es cuando una transacción T_1 se interesa en escritura, en el mismo objeto que se interesa T_2 , pero esta última lo hace en lectura.

En este caso se puede presentar una lectura indebida de un objeto, como se aprecia en el siguiente ejemplo:

Ejemplo. En la figura 9.4 se muestra un caso de dos transacciones T_1 y T_2 , en donde T_2 ha leído un valor actualizado por T_1 , lo ha escrito y T_1 , debido por ejemplo, a una falla del sistema, ha tenido que ser eliminada. Así, no debe quedar rastro de las modificaciones hechas por T_1 ; sin embargo, T_2 utilizó un valor modificado por T_1 .

Por lo tanto, para evitar una lectura indebida, una transacción no debe leer y modificar valores que hayan sido modificados por otra transacción cuando esta aun no ha sido validada, es decir no ha alcanzado el punto de **commit**.

T ₁	T ₂	valor real	valor esperado
begin T₁			
read a	begin T₂	a = 1000	a = 1000
a = a+2000		a = 3000	a = 3000
write a	read a	a = 3000	a = 3000
	read c	c = 5000	c = 5000
	c = a + 1000		
	write c	c = 4000	c = 4000
	commit T₂		
abort T₁			

Figura 9.4 Ejemplo de lectura indebida

9.3.3 Lectura-escritura

En el siguiente ejemplo se aprecia un problema que puede darse cuando una transacción T₁ se interesa en lectura y otra transacción T₂ en escritura sobre el mismo objeto, y en donde la segunda, sin modificar el objeto y al querer leer más de una vez este objeto, esperaría contar con el mismo valor.

Ejemplo. En el caso de la figura 9.5, la transacción T₂ ha leído dos veces el valor de a.

T ₁	T ₂	valor real	valor esperado
begin T₁			
read a	begin T₂	a = 1000	a = 1000 (T ₁)
	read a	a = 1000	a = 1000 (T ₂)
a = a+2000			
write a		a = 3000	a = 3000 (T ₁)
commit T₁	read a	a = 3000	a = 1000 (T ₂)

Figura 9.5 Ejemplo de lecturas no concordantes

valor esperado
a = 1000
a = 3000
a = 3000
c = 5000
c = 4000

ndebida

problema que puede darse
ura y otra transacción T_2 en
la segunda, sin modificar el
objeto, esperaría contar con

ón T_2 ha leído dos veces el valor

valor esperado
a = 1000 (T_1)
a = 1000 (T_2)
a = 3000 (T_1)
a = 1000 (T_2)

o concordante's

Sin embargo, T_2 no lo ha modificado. Entonces, se esperaría que T_2 pudiera leer el mismo valor.

Así, una transacción no debe modificar un valor leído por otra, antes de que esta última no haya terminado.

Una vez que se han visto los diferentes conflictos que se pueden presentar al realizar concurrentemente varias transacciones, se debe introducir el concepto de itinerario.

9.4 ITINERARIO

Cuando varias transacciones se realizan en forma concurrente, se dice que se tiene un *itinerario* -en inglés los términos utilizados son *schedule*, *history* o *audit*-.

Por otra parte, el programa que controla la ejecución concurrente de las transacciones se denomina *itinerador* -del inglés *scheduler*-.

Para ejecutar una operación, la transacción involucrada envía tal operación al itinerador. Luego de recibir dicha operación, este programa realiza alguna de las acciones siguientes: ejecutar, rechazar o dejar en espera la operación.

Como se estudió en la sección anterior, varios problemas se pueden presentar cuando se quieren realizar transacciones concurrentemente.

Formalmente, se puede decir que un itinerario I sobre n transacciones T_1, T_2, \dots, T_n , se puede definir como una sucesión de operaciones

$$I = (x_1y_1\dots u_1) (x_2y_2\dots u_2) \dots (x_p y_p \dots u_p)$$

en donde

x_i es un conjunto de 0 o más operaciones de T_1

y_i es un conjunto de 0 o más operaciones de T_2

...

u_i es un conjunto de 0 o más operaciones de T_n

y además que

$$x_1 x_2 x_3 \dots x_p = T_1$$

$$y_1 y_2 y_3 \dots y_p = T_2$$

...

$$u_1 u_2 u_3 \dots u_p = T_n$$

Ejemplo. Sean las siguientes transacciones

T_1 :	read a write a	T_2 :	read b write b
---------	-------------------	---------	-------------------

Entonces, en este caso se tienen seis posibles itinerarios para las transacciones T_1 y T_2 , según se aprecia en la figura 9.6.

I_1	I_2	I_3	I_4	I_5	I_6
read a	read b	read a	read b	read b	read a
write a	write b	read b	read a	read a	read b
read b	read a	write a	write a	write b	write b
write b	write a	write b	write b	write a	write a

Figura 9.6 Ejemplo de itinerarios de dos transacciones

Es fácil ver que los itinerarios secuenciales siempre darán resultados consistentes y no tendrán conflictos de concurrencia. Es el caso de los itinerarios I_1 y I_2 del ejemplo. Los otros cuatro no son secuenciales.

Así, la idea es contar con herramientas que permitan discernir, de los itinerarios no secuenciales, aquellos que son *consistentes*, es decir, aquellos que sean equivalentes a uno secuencial y por lo tanto que den los mismos resultados.

Ejemplo. Considerar tres itinerarios I_1 , I_2 y I_3 de dos transacciones T_1 y T_2 , las cuales rebajan respectivamente 1000 colones de las cuentas a y c y los acredita a la cuenta b, según se aprecia en la figura 9.7.

En este caso, I_1 es un itinerario secuencial pues primero se realiza la transacción T_1 y luego la transacción T_2 y por lo tanto es consistente. Por su parte, I_2 no es

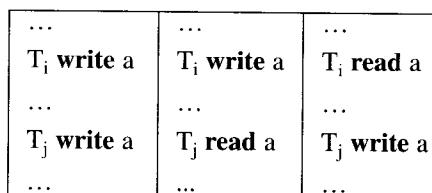
secuencial, sin embargo, sí es consistente, pues da los mismos resultados que I_1 . Finalmente, I_3 no es ni secuencial ni consistente.

I_1	valor real	I_2	valor real	I_3	valor real
T1: begin		T1: begin		T1: begin	
T1: read a	a=2000	T1: read a	a=2000	T1: read a	a=2000
T1: read b	b=3000	T2: begin		T2: begin	
T1: a=a-1000		T2: read b	b=3000	T1: read b	b=3000
T1: b=b+1000		T2: read c	c=4000	T2: read c	c=4000
T1: write a	a=1000	T1: a=a-1000		T1: a=a-1000	
T1: write b	b=4000	T1: write a	a=1000	T2: read b	b=3000
T1: commit		T2: b=b+1000		T1: b=b+1000	
T2: begin		T2: c=c-1000		T2: c=c-1000	
T2: read c	c=4000	T2: write b	b=4000	T1: write a	a=1000
T2: read b	b=4000	T2: write c	c=3000	T2: b=b+1000	b=4000
T2: c=c-1000		T1: read b	b=4000	T1: write b	b=4000
T2: b=b+1000		T2: commit		T2: write c	c=3000
T2: write c	c=3000	T1: b=b+1000		T1: commit	
T2: write b	b=5000	T1: write b	b=5000	T2: write b	
T2: commit		T1: commit		T2: commit	

Figura 9.7 Ejemplo de itinerarios

Una forma en que un SABD puede determinar si un itinerario no secuencial es equivalente a uno secuencial es por medio del *grafo de dependencias* [DELO82]. En este grafo los nodos representan las diferentes transacciones que conforman el itinerario. Por su parte, los arcos de dicho grafo se establecen de la siguiente forma:

Suponer que dos transacciones T_i y T_j se interesan sobre el mismo objeto a y se da alguna de las condiciones siguientes:



Entonces habrá una flecha de T_i hacia T_j :

$$T_i \rightarrow T_j.$$

En un itinerario secuencial el grafo de dependencias asociado no contiene ciclos pues si se tiene el arco $T_i \rightarrow T_j$, entonces T_i precede a T_j en el itinerario. Así, para que un itinerario sea consistente -equivalente a uno secuencial- basta que el grafo de dependencias asociado no contenga ciclos.

Ejemplo. En la figura 9.8 aparecen los grafos de dependencia de los itinerarios del ejemplo de la figura 9.7.

Considerando el objeto b, en este caso se tiene que los itinerarios I_1 y I_2 son consistentes y como el grafo asociado con I_3 posee un ciclo, no es consistente, es decir, no es equivalente a uno secuencial.

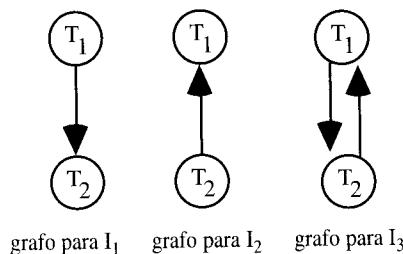


Figura 9.8 Grafos de dependencia de tres itinerarios

9.5 MECANISMOS PARA CONTROLAR LA CONCURRENCIA

Una forma de solucionar el problema de los conflictos en la concurrencia es por medio de la definición de *cerrojos* sobre los objetos que pueden generar dichos conflictos. Un cerrojo es una operación que permite a una transacción tomar el control del objeto deseado, en esta situación, y otra transacción puede accesarlo solo si la transacción inicial lo libera.

Estas dos operaciones se denotarán por las palabras en inglés

lock
unlock

dependencias asociado no T_j , entonces T_i precede a T_j
sea consistente -equivalente a
encias asociado no contenga

dependencia de los itinerarios del
que los itinerarios I_1 y I_2 son
un ciclo, no es consistente, es



afo para I_3

e tres itinerarios

R LA CONCURRENCIA

a de los conflictos en la
de cerrojos sobre los objetos
cerrojo es una operación que
del objeto deseado, en esta
solo si la transacción inicial

as palabras en inglés

Un conjunto de reglas que deben ser verificadas por varias transacciones cuando se realizan concurrentemente con el fin de establecer un itinerario consistente es lo que se denomina *protocolo*.

Un protocolo clásico para la administración de recursos que utiliza las dos operaciones de bloqueo anteriores se denomina *protocolo de exclusión mutua*, el cual se presenta a continuación [DELO82].

Protocolo de exclusión mutua

1. Ninguna transacción puede efectuar una actualización o una lectura de un objeto a si no ha adquirido previamente un cerrojo sobre él.
2. Si una transacción no puede adquirir un cerrojo de un objeto a debido a que está siendo utilizado por otra transacción, entonces la segunda debe esperar hasta que sea liberado por la primera.

Ejemplo. Si se considera el itinerario de la figura 9.9, las dos transacciones que conforman dicho itinerario se realizan concurrentemente siguiendo el protocolo de exclusión mutua.

I	valor real	valor esperado
T1: begin		
T1: lock a		
T1: read a	a = 1000	a = 1000
T2: lock a		
T1: a = a+3000	a = 4000	a = 4000
T2: wait		
T1: write a	a = 4000	a = 4000
T2: wait		
T1: unlock a		
T2: wait		
T1: commit		
T2: read a	a = 4000	a = 4000
T2: a = a+6000		
T2: write a	a = 10000	a = 10000
T2: unlock a		
T2: commit		

Figura 9.9 Itinerario según el protocolo de exclusión mutua

Un problema que se puede presentar cuando dos o más transacciones se interesan cada una en un objeto que ha sido bloqueado por la otra y en donde se cae en una espera indefinida por parte de las transacciones involucradas es el llamado *interbloqueo* -del inglés *deadlock*-.

Para comprender este concepto, se introduce el siguiente ejemplo.

Ejemplo. Sea el itinerario I conformado por dos transacciones según se aprecia en la figura 9.10.

I
T ₁ : begin
T ₁ : lock a
T ₂ : begin
T ₂ : lock b
T ₁ : lock b
T ₂ : lock a
T ₁ : wait
T ₂ : wait
T ₁ : wait
T ₂ : wait
T ₁ : wait
...

Figura 9.10 Ejemplo de interbloqueo

En este caso, la transacción T₁ ha adquirido un cerrojo sobre el objeto *a*. Posteriormente T₂ adquiere el control del objeto *b*. Luego, T₁ se interesa en el objeto *b* y T₂ en el objeto *a*, sin haber sido liberados por ninguna de las dos transacciones. Así, se entra en una espera indefinida por parte de las dos transacciones.

Una forma de resolver el problema del interbloqueo es por medio de una asignación de un número único a cada transacción llamado *estampilla de tiempo*. Usando estampillas se pueden ordenar las solicitudes de las transacciones en forma completa. Existen dos opciones en la asignación de estampillas. Por una parte, se puede hablar del

método
espera
recien
con de
es dec
estamp
a fin d

A c
es muc
en la m

9.5.1 P

Con
cerrojo
el cerro

El ce
inglés s
cerrojo

Esto
lectura p

Por s
x-lock-
adquirid
escribir s

En la
cerrojos

do dos o más transacciones
ido bloqueado por la otra y
parte de las transacciones
inglés *deadlock*.

uce el siguiente ejemplo.

transacciones según se aprecia en

bloqueo

jo sobre el objeto *a*. Posteriormente,
eresa en el objeto *b* y T_2 en el objeto
transacciones. Así, se entra en una

nterbloqueo es por medio de
cada transacción llamado
as se pueden ordenar las
mpleta. Existen dos opciones
parte, se puede hablar del

método *sin decomiso* -en inglés *wait die*- que obliga a una transacción a esperar si entra en conflicto con otra transacción cuya estampilla es más reciente o a abortar en caso contrario. El otro mecanismo es el llamado *con decomiso* -en inglés *wound wait*- el cual procede en forma inversa, es decir, en caso de que una transacción entra en conflicto con otra cuya estampilla es más reciente obliga a ésta última a esperar, o bien, a abortar a fin de tomar el objeto en cuestión.

A continuación, se va a estudiar el protocolo llamado de dos fases que es mucho más eficiente que el anterior y es el que ha sido implementado en la mayoría de los SABDs comercializados.

9.5.1 Protocolo de dos fases

Con el protocolo de dos fases, se afina aún más el concepto de cerrojo. En efecto, se usan dos tipos de cerrojo: el cerrojo compartido y el cerrojo exclusivo.

El *cerrojo de lectura* o *compartido* que se denota por **s-lock**- del inglés *share lock*-, permite a una transacción que ha adquirido dicho cerrojo sobre un objeto, únicamente leer dicho objeto.

Esto conduce al hecho de que un objeto puede ser compartido en lectura por varias transacciones concurrentes.

Por su parte, el *cerrojo exclusivo* o de *escritura* que se escribe como **x-lock**- del inglés *exclusive lock*-, permite a una transacción que ha adquirido dicho cerrojo sobre un objeto, no solo leerlo sino también escribir sobre él.

En la figura 9.11 se presenta una tabla de compatibilidad de estos cerrojos en transacciones concurrentes.

Cerrojo	s-lock	x-lock
s-lock	compatible	conflicto
x-lock	conflicto	conflicto

Figura 9.11 Compatibilidad de cerrojos

Así, se dice que existe conflicto entre dos cerrojos si y solamente si:

- a. Conciernen el mismo objeto
- b. Un cerrojo es **x-lock** y el otro es **x-lock o s-lock**.

Por su parte, se dice que una transacción T se encuentra *bien formada* si y solamente si satisface las siguientes propiedades:

1. Para que una transacción T pueda leer un objeto *a*, previamente debe adquirir un cerrojo **s-lock** a.
2. Para que una transacción T pueda escribir sobre un objeto *a*, previamente debe adquirir un cerrojo **x-lock** a.
3. Despues de que la transacción T haya llegado a su punto de validación, ningún objeto queda bloqueado.

Ejemplo. Sean las dos transacciones siguientes:

begin transaction T₁	begin transaction T₂
s-lock a	x-lock a
read a	read a
a = a-1000	a = a-5000
write a	write a
unlock a	x-lock b
x-lock b	unlock a
read b	read b
b = b+1000	b = b+5000
write b	write b
commit T₁	unlock b
	commit T₂

En este caso se tiene que la transacción T₁ no está bien formada puesto que escribió sobre el objeto *a* habiendo adquirido solo un cerrojo compartido y además concluyó sin haber liberado el cerrojo exclusivo que tenía sobre el objeto *b*. Por su parte, la transacción T₂ sí está bien formada.

A partir de este momento solo se consideran transacciones bien formadas y a continuación se enuncian las dos reglas que establecen el llamado protocolo de dos fases.

s cerrojos si y solamente si:

ck o s-lock.

T se encuentra *bien formada* si cumple las siguientes propiedades:

er un objeto a , previamente

escribir sobre un objeto a , no se adquiere el **x-lock** a.

maya llegado a su punto de bloqueo.

ction T_2

0

0

á bien formada puesto que escribió un cerrojo compartido y además concluyó su escritura sobre el objeto b . Por su parte, la

onsideran transacciones bien formadas si cumplen las dos reglas que establecen el

Protocolo de dos fases

1. Dos transacciones no pueden tener cerrojos en conflicto en forma simultánea.
2. Toda transacción que libera un cerrojo no puede adquirir otro.

Según esto, como el nombre lo dice, se tienen dos fases: una de *expansión* que es cuando se adquieren los cerrojos y otra de *reducción* que es cuando se liberan los cerrojos, según se aprecia en la figura 9.12 [GARD85a].

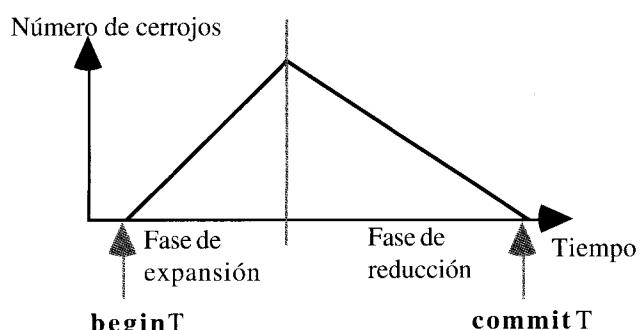


Figura 9.12 Evolución de una transacción de dos fases

Existe un resultado muy importante y es el hecho de que si en un itinerario todas las transacciones se encuentran en dos fases, entonces el itinerario es equivalente a un itinerario secuencial. Una demostración de este resultado se puede encontrar en [DELO82].

Ejemplo. Sean los itinerarios descritos en la Figura 9.13:

En este caso se tiene que I_1 no es consistente pues no satisface el protocolo de dos fases. Por su parte el itinerario I_2 sí es consistente.

I_1	I_2 :
T ₁ : begin	T ₃ : begin
T ₁ : x-lock a	T ₃ : x-lock a
T ₁ : read a	T ₃ : read a
T ₂ : begin	T ₃ : a = a-5000
T ₂ : x-lock b	T ₃ : write a
T ₂ : read b	T ₃ : x-lock b
T ₁ : a = a-1000	T ₃ : read b
T ₂ : b = b+1000	T ₃ : unlock a
T ₁ : write a	T ₄ : begin
T ₂ : write b	T ₄ : s-lock a
T ₁ : unlock a	T ₃ : b = b+5000
T ₂ : unlock b	T ₃ : write b
T ₁ : s-lock b	T ₃ : unlock b
T ₁ : read b	T ₃ : commit
T ₁ : b = b+2000	T ₄ : s-lock b
T ₁ : unlock b	T ₄ : print a*b
T ₂ : commit	T ₄ : unlock a
T ₁ : commit	T ₄ : unlock b
	T ₄ : commit

Figura 9.13 Ejemplo de dos itinerarios

9.5.2 Organización jerárquica de los objetos de la base de datos

El protocolo de dos fases introducido en la sección anterior establece cerrojos sobre datos individuales. Puede producirse la generalización de estos cerrojos, es decir, establecer unidades de bloqueo que permitan no solo bloquear tuplas, sino también tablas, segmentos o incluso toda la base de datos. El trabajo pionero en este sentido se puede encontrar en [GRAY75].

Este enfoque de bloqueo de conjuntos de objetos requiere agrupar los objetos por niveles. De hecho, en el marco de la organización física de una base de datos, se tienen varios niveles, según se aprecia en la figura 9.14.

I₂:
begin
lock a
read a
= a-5000
rite a
lock b
read b
unlock a
begin
lock a
= b+5000
rite b
unlock b
commit
lock b
print a*b
unlock a
unlock b
commit
itinerarios

Objetos de la base de datos

dido en la sección anterior
tales. Puede producirse la
cir, establecer unidades de
tuplas, sino también tablas,
s. El trabajo pionero en este

de objetos requiere agrupar
arco de la organización física
veles, según se aprecia en la

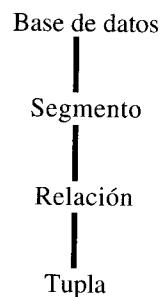


Figura 9.14 Niveles de jerarquía en una base de datos

En tales circunstancias se pueden considerar dos enfoques. En primer lugar se tiene la posibilidad de colocar un cerrojo en un objeto sin afectar a sus descendientes. La otra opción es colocar un cerrojo en un objeto bloqueando implícitamente a sus descendientes. A continuación se estudian dos protocolos en un ambiente de jerarquía.

9.5.3 Protocolo de árbol

El protocolo de árbol puede aplicarse si se tiene información sobre la forma en que se acceden los objetos, particularmente cuando se puede establecer un orden parcial en el acceso a estos objetos y esto puede verse como un árbol con una raíz.

A continuación, se presenta dicho protocolo [KORT86] el cual solo requiere cerrojos exclusivos.

Protocolo de árbol

1. El primer cerrojo de una transacción puede hacerse en cualquier objeto.
2. Una transacción puede bloquear un objeto si el padre del mismo -en el árbol- posee un cerrojo.
3. Los objetos se liberan en cualquier momento.
4. Una vez que una transacción ha liberado un objeto, no puede bloquearlo nuevamente.

Se dice que un itinerario es consistente, cuando las transacciones involucradas cumplen con el protocolo de árbol.

Ejemplo. En la figura 9.15 se muestra un itinerario donde las transacciones cumplen con el protocolo de árbol.

I
T ₁ : begin
T ₁ : lock a
T ₁ : read a
T ₁ : a = a+c
T ₁ : write a
T ₁ : unlock a
T ₂ : begin
T ₂ : lock a
T ₂ : read a
T ₂ : a = a+d
T ₂ : write a
T ₁ : lock b
T ₁ : read b
T ₁ : b = b+c
T ₁ : write b
T ₁ : unlock b
T ₁ : end
T ₂ : lock b
T ₂ : read b
T ₂ : b = b+f
T ₂ : write b
T ₂ : unlock a
T ₂ : unlock b
T ₂ : end

Figura 9.15 Ejemplo de un itinerario cuyas transacciones satisfacen el protocolo de árbol

e, cuando las transacciones
rbol.

inerario donde las transacciones

En el ejemplo anterior se puede ver que las transacciones no requieren estar en dos fases. Así, los objetos pueden ser liberados antes, lo que puede repercutir en un tiempo de espera menor y en un mejoramiento de la concurrencia.

9.5.4 Protocolo de granularidad múltiple

Si una transacción requiere bloquear toda la base de datos, sería mejor que se pusiera un solo cerrojo a la base de datos y no a cada uno de los elementos que componen dicha base de datos. Lo mismo se podría decir de una relación. En este caso si se desea bloquear las tuplas de una relación, se debería bloquear solo la relación y que las tuplas que la componen automáticamente queden bloqueadas. Además, el sistema debe saber que en la base de datos hay un segmento, en donde se encuentra una relación bloqueada.

Bajo este contexto, se debe establecer el nivel del objeto que se desea bloquear el cual se conoce como *gránulo*. Este puede variar desde la base de datos completa hasta el valor de un atributo.

Ejemplo. Sea la base de datos de la figura 9.16. En este caso se tiene una base de datos BD_1 que se almacena en dos segmentos S_1 y S_2 . Además, el segmento S_1 contiene la relación (o archivo) R_1 que a su vez, contiene la tupla t_1 . Por su parte, el segmento S_2 contiene las relaciones R_2 y R_3 , que a su vez, contienen las tuplas t_2, t_3, t_4 y t_5 .

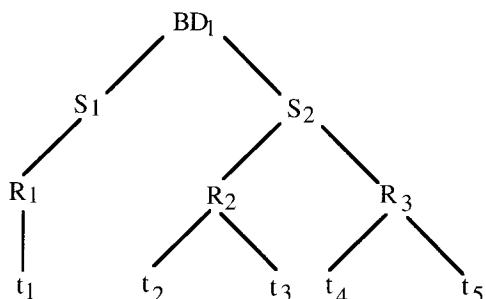


Figura 9.16 Ejemplo de almacenamiento de relaciones en segmentos

transacciones satisfacen

ol

Así, cada nodo de la jerarquía de la organización física puede ser bloqueado. Esto significa que si se pone un cerrojo exclusivo en un nodo cualquiera, implícitamente debería garantizarse el cerrojo en todos los descendientes de dicho nodo. Lo mismo se podría decir de un cerrojo compartido. Entonces, el cerrojo que se ponga a un nodo, estaría bloqueando implícitamente el sub-árbol cuya raíz es precisamente el nodo bloqueado.

Pero para que el bloqueo de un sub-árbol se pueda dar es necesario identificar los ancestros del nodo que se desea bloquear con el fin de prevenir un cerrojo exclusivo o compartido de los mismos.

Por esta razón se introduce un nuevo tipo de cerrojo llamado *de intención* [GRAY75]. Este tipo de cerrojo sirve para marcar los ancestros de un nodo al que se quiere aplicar un cerrojo exclusivo o compartido.

Para refinar aun más la compatibilidad de cerrojos se establece entonces un cerrojo de lectura en intención y otro de escritura en intención y se denominan respectivamente

is-lock
ix-lock

-del inglés *intention shared lock* e *intention exclusive lock* respectivamente-.

Este refinamiento se da debido a dos razones [GRAY75]. En primer lugar el cerrojo **is-lock** solo solicita cerrojos **s-lock** e **is-lock** en los niveles inferiores del nodo. Como el acceso de solo lectura es muy común, es beneficioso distinguir este tipo de acceso. En segundo lugar, si una transacción tiene un **is-lock** sobre un nodo, posteriormente se puede reemplazar por un cerrojo **s-lock**; sin embargo esta situación no se mantiene si se desea convertir un cerrojo **ix-lock** en un cerrojo **s-lock**.

Por otra parte, ¿qué ocurre si se desea leer un sub-árbol y modificar solo algunos nodos de éste? Se tienen dos posibilidades para responder a esto:

a. C
b. C
c.
d.

La p
nodos c
concorre
número

La s
permite
nodos s
los nod
exclusiv
en el res

Por e

-del inglés
lecturas

En la
cerrojos-

Cerre
s-lock
x-lock
is-lock
ix-lock
six-lock

organización física puede ser un cerrojo exclusivo en un nodo. Colocarse el cerrojo en todos los nodos se podría decir de un cerrojo que pone a un nodo, estaría cuya raíz es precisamente el

subárbol se pueda dar es necesario que se desea bloquear con el fin de control de los mismos.

Otro tipo de cerrojo llamado de escritura sirve para marcar los ancestros de un cerrojo exclusivo o compartido.

La cantidad de cerrojos se establece uno de lectura y otro de escritura en

intention exclusive lock

los razones [GRAY75]. En una consulta se solicita cerrojos **s-lock** e **is-lock**. Si el acceso de solo lectura es el tipo de acceso. En segundo lugar se solicita un cerrojo **is-lock** sobre un nodo, ya sea por un cerrojo **s-lock**; si no, si se desea convertir un

se lee un subárbol y modificar las posibilidades para responder

- Colocar un cerrojo **x-lock** en el nodo raíz del subárbol
- Colocar un cerrojo **ix-lock** en el nodo raíz y luego poner un cerrojo explícito -exclusivo, compartido o en intención- en los descendientes del nodo.

La primera opción no es recomendable pues bloquea incluso los nodos del subárbol que no serán utilizados y por ende no existe concurrencia. Por su parte, la segunda posibilidad conduce a solicitar un número grande de cerrojos.

La solución a este problema sería contar con un cerrojo que permitiera compartir el subárbol de forma tal que se puedan leer los nodos sin necesidad de solicitar más cerrojos y compartir en intención los nodos del subárbol de forma que se puedan colocar cerrojos exclusivos en los nodos que se desean modificar y colocar otros cerrojos en el resto de los nodos del subárbol.

Por esta razón se introduce el tipo de cerrojo

six-lock

-del inglés *shared and intention exclusive lock*-, que va a permitir hacer lecturas y modificaciones en el subárbol.

En la figura 9.17 se presenta una tabla de compatibilidades de los cerrojos anteriormente definidos.

Cerrojo	s-lock	x-lock	is-lock	ix-lock	six-lock
s-lock	compatible	conflicto	compatible	conflicto	conflicto
x-lock	conflicto	conflicto	conflicto	conflicto	conflicto
is-lock	compatible	conflicto	compatible	compatible	compatible
ix-lock	conflicto	conflicto	compatible	compatible	conflicto
six-lock	conflicto	conflicto	compatible	conflicto	conflicto

Figura 9.17 Compatibilidad de cerrojos

A continuación, se presenta el protocolo de *granularidad múltiple* debido a Gray [GRAY78] que permite determinar si un itinerario, en donde los objetos se organizan jerárquicamente, es o no consistente.

Protocolo de bloqueo jerárquico

1. Para que una transacción adquiera un cerrojo **s-lock** o **is-lock** sobre un objeto, debe adquirir previamente un cerrojo **is-lock** o un **ix-lock** sobre todos los ancestros de este objeto.
2. Para que una transacción adquiera un cerrojo **x-lock**, **ix-lock** o un **six-lock**, debe adquirir previamente un cerrojo **ix-lock** o un **six-lock** sobre todos los ancestros de este objeto.
3. Una transacción no puede poner un cerrojo a un objeto si previamente ha liberado algún nodo.
4. Una transacción puede liberar el cerrojo sobre un objeto solo si ninguno de sus descendientes tiene un cerrojo puesto por la transacción.

El protocolo de bloqueo jerárquico requiere que los cerrojos se pongan en orden descendente, es decir, de la raíz hacia las hojas y se liberen en orden ascendente, es decir, desde las hojas hacia la raíz.

Ejemplo. Sea el árbol de la figura 9.19. Suponga que una transacción T, solicita leer la tupla t_3 . Entonces se debe hacer:

is-lock	BD_1
is-lock	S_2
is-lock	R_2
s-lock	t_3

Si se desea modificar la tupla t_1 , entonces se debe hacer:

ix-lock	BD_1
ix-lock	S_1
ix-lock	R_1
x-lock	t_1

lo de *granularidad múltiple* terminar si un itinerario, en efecto, es o no consistente.

árquico

sobre un cerrojo **s-lock** o **is-lock**

sobre un **x-lock**, **ix-lock** o un **six-lock**

a un objeto si previamente

sobre un objeto solo si un cerrojo puesto por la

requiere que los cerrojos se de la raíz hacia las hojas y se de las hojas hacia la raíz.

que una transacción T, solicita leer

be hacer:

Si se desea leer la relación R_3 para modificar las tuplas t_3 y t_4 , se debe hacer:

ix-lock	BD_1
ix-lock	S_2
six-lock	R_3
x-lock	t_3
x-lock	t_4

9.6 TRANSACCIONES COMPLEJAS Y DE LARGA DURACION

En las anteriores secciones se introdujo el concepto de transacción como un conjunto de operaciones o acciones mediante el cual un estado consistente de la base de datos es transformado en otro estado consistente.

Estas transacciones han sido, por lo general, de tipo administrativo y relativamente cortas en el tiempo.

Sin embargo, con el advenimiento de nuevas aplicaciones en ambientes de bases de datos -ingeniería de software, aplicaciones CAD/CAM, descripción del código genético de una célula, etc.-, se requiere el manejo de transacciones cada vez más complejas y de larga duración -varias horas o días en realizarse- y los mecanismos utilizados hasta hoy se vuelven caducos.

Debido a la incapacidad de que un ambiente actual de bases de datos maneje este tipo de transacciones, varios autores, entre otros, [BANC85], [YEH 87], coinciden en los siguientes requerimientos mínimos para poder soportar este tipo de transacciones [BARG91]:

- *Reevaluación de las políticas de reducción.* Cuando se tienen transacciones que duran varios días en ejecutarse, no es posible, ante una caída del sistema, que la transacción deba eliminarse si ésta aun no ha llegado a su punto de validación. Tampoco es admisible que una transacción de larga duración se ponga en espera hasta que otra termine.

- *Control del usuario en las transacciones.* Cuando se realizan transacciones complejas y/o de larga duración, es posible que se requiera, sobre la marcha, el uso de varios recursos como código fuente, bibliotecas, datos de prueba, documentación, etc., y esto en forma aleatoria, lo cual no coincide necesariamente con el esquema usual de un SABD.
- *Soporte para la cooperación en línea.* Es importante contar con mecanismos que permitan soportar el trabajo cooperativo en forma interactiva, como es el caso del diseño CAD en el cual varias personas puedan requerir trabajar en forma concurrente con objetos comunes.

Debido a estas situaciones, se han hecho varias propuestas.

Con respecto al protocolo de dos fases, éste es inaceptable en el manejo de la concurrencia de transacciones de larga duración debido a que obligaría a bloquear los objetos durante un largo período de tiempo, incluso luego de haber concluido la utilización de estos objetos. Esto obligaría a una repetición de borrar las acciones de las transacciones pues cuando se tienen transacciones de este tipo crecen los conflictos.

Una posibilidad que se sugiere en [BARG91] para resolver este tipo de problema es extraer información semántica de las transacciones y las operaciones y usar este tipo de información para adaptar las técnicas convencionales. Una de estas soluciones es el llamado *bloqueo generoso* -del inglés *altruistic locking*-. Este mecanismo es una extensión del mecanismo de dos fases y usa información sobre los patrones de acceso de una transacción con el fin de decidir cuáles objetos pueden liberarse.

La información es de dos tipos:

- *acceso negativo:* que describe los objetos que no pueden ser accesados.
- *acceso positivo:* el cual describe el orden y cuáles objetos pueden ser accesados por una transacción.

aciones. Cuando se realizan duración, es posible que se varios recursos como código documentación, etc., y esto en necesariamente con el esquema

a. Es importante contar con trabajo cooperativo en forma diseño CAD en el cual varias forma concurrente con objetos

o varias propuestas.

es. éste es inaceptable en el de larga duración debido a un largo período de tiempo. zación de estos objetos. Esto acciones de las transacciones tipo crecen los conflictos.

RG91] para resolver este tipo tica de las transacciones y las ión para adaptar las técnicas el llamado *bloqueo generoso* nismo es una extensión del sobre los patrones de acceso illes objetos pueden liberarse.

objetos que no pueden ser

den y cuáles objetos pueden

Al combinar estos tipos de información, es posible maximizar el aprovechamiento de los objetos cuando se realizan las transacciones de larga duración.

La liberación de un objeto se maneja en forma condicional; de esta forma se permite que varias transacciones lo accesen, siempre y cuando se cumpla la consistencia del itinerario.

Así, el protocolo de dos fases adaptado a tal ambiente, se hace agregando las dos reglas siguientes:

- Dos transacciones no pueden bloquear el mismo objeto mientras traslanan, a menos que una tenga un cerrojo y libere el objeto antes que otro lo bloquee, el último cerrojo preservado se dice que está en el despertar - en inglés *wake-* de la transacción liberada.
- Si una transacción está en el despertar de otra transacción, ésta debe completarse en el despertar de la primera transacción.

Por otra parte, se tiene el concepto de sagas desarrollado en [GARC87] que sugiere una subdivisión de las transacciones de larga duración que se pueden intermezclar con otras y promete un mejoramiento en el rendimiento del sistema en estos casos.

Una *saga* es una transacción de larga duración que puede fraccionarse en un conjunto de sub-transacciones que se pueden intercalar con otras transacciones, de manera que estas sub-transacciones formen una unidad en la realización, guardando cada una de ellas la consistencia de la base de datos.

En caso de que la transacción no se pueda realizar completamente, se deben realizar acciones compensatorias. Una *operación compensatoria* es una transacción que deshace las acciones realizadas por la sub-transacción respectiva pero no necesariamente deja la base de datos en el estado en que se encontraba antes de realizarse esta sub-transacción.

Formalmente, una saga es una transacción de larga duración que se compone de un conjunto de transacciones $T = \{T_1, T_2, \dots, T_n\}$ y un

conjunto de acciones compensatorias $C = \{C_1, C_2, \dots, C_{n-1}\}$ a estas transacciones.

El sistema debe garantizar que se ejecute una de las siguientes dos sucesiones:

- $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n$
- $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_j \rightarrow C_j \rightarrow \dots \rightarrow C_2 \rightarrow C_1$, con $0 \leq j < n$.

Las otras transacciones pueden conocer los efectos de ejecuciones parciales de una saga. Si ésta aborta, no existe interferencia con el resto de las transacciones, de hecho no se les informa de esta operación.

Para comprender mejor este concepto se utilizará el siguiente ejemplo presentado en [GARC87].

Ejemplo. Considerar una aplicación de reservaciones de asientos en varias líneas aéreas. Suponer que una transacción T que realiza varias reservaciones se considera como una saga. En esta aplicación puede que no se requiera que T tenga el control de los recursos hasta que ésta se confirme. Así, luego de que T reserva un asiento del vuelo V_1 , podría permitir que otras transacciones reserven asientos en el vuelo V_1 , es decir, que T puede verse como un conjunto de sub-transacciones T_1, T_2, \dots, T_n que van a reservar asientos en forma individual.

Sin embargo, no se desea realizar la transacción T como un conjunto de transacciones independientes pues lo que se desea es que se realice o que no se haga del todo.

Por otra parte, recordar que cada sub-transacción T_i tiene asociada una transacción de compensación C_i . Así, la transacción de compensación deshace, desde un punto de vista semántico, cualquiera de las transacciones realizadas por T_i , pero no se devuelve necesariamente al estado cuando se inició T_i . Así, si T_i reserva un asiento de un vuelo V_1 , C_i puede cancelar la reservación. Pero la transacción de compensación C_i no puede simplemente almacenar en la base de datos el número de asientos que existían cuando se ejecutó T_i ya que otras podrían haberse ejecutado en el tiempo que transcurre desde que T_i reservó el asiento y C_i canceló la reservación.

EJERCICIOS Y PREGUNTAS DE REPASO

9.1 Definir el concepto de transacción.

9.2

9.3

9.4

9.5

9.6

9.7

9.8

9.9

D

C

C

$\{C_1, C_2, \dots, C_{n-1}\}$ a estas

e una de las siguientes dos

, con $0 \leq j < n$.

los efectos de ejecuciones
ste interferencia con el resto
rma de esta operación.

utilizará el siguiente ejemplo

iones de asientos en varias líneas
varias reservaciones se considera
se requiera que T tenga el control
uego de que T reserva un asiento
nes reserven asientos en el vuelo
o de sub-transacciones T_1, T_2, \dots
l.

como un conjunto de transacciones
e o que no se haga del todo.
, tiene asociada una transacción de
ación deshace, desde un punto de
lizadas por T_i , pero no se devuelve
si T_i reserva un asiento de un vuelo
nsacción de compensación C_i no
el número de asientos que existió
erse ejecutado en el tiempo que
celó la reservación.

9.2 ¿Cuándo se tiene un conflicto entre transacciones? Clasificar dichos conflictos.

9.3 Definir el concepto de itinerario y de itinerario consistente.

9.4 Explicar los diferentes tipos de cerrojos.

9.5 Enunciar el protocolo de dos fases.

9.6 ¿Qué es un gránulo?

9.7 Explicar la noción de sagas.

9.8 Considere el siguiente itinerario de las transacciones T_1 y T_2 :

$T_1: begin$

$T_1: x-lock c$

$T_1: read c$

$T_2: begin$

$T_2: x-lock d$

$T_2: read d$

$T_1: c = c - 1000$

$T_2: d = d + 1000$

$T_1: write c$

$T_2: write d$

$T_1: unlock c$

$T_2: unlock d$

$T_2: commit$

$T_1: s-lock d$

$T_1: read d$

$T_1: d = d + 2000$

$T_1: unlock d$

$T_1: commit$

¿Es el itinerario anterior consistente? Justifique su respuesta.

9.9. [KORT86] Considere un SABD que incluye, además de las operaciones **read** y **write**, la operación **increment**. Sea v el valor del objeto x . La operación **increment** x en c asigna el valor $v+c$ a

x en un paso atómico. La transacción no dispondrá del valor de x mientras no ejecute un **read x**. La siguiente figura muestra la compatibilidad de cerrojos para tres modos de cerrojo: compartido, exclusivo y de incrementación.

	s-lock	x-lock	increment
s-lock	compatible	conflicto	conflicto
x-lock	conflicto	conflicto	conflicto
increment	conflicto	conflicto	compatible

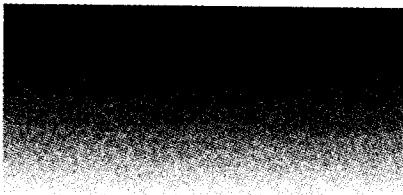
Demostrar que si todas las transacciones ponen cerrojos en el modo correspondiente a los datos que accesan, el protocolo de dos fases garantiza la consistencia.

Demostrar que el modo **increment** permite mayor concurrencia.

o dispondrá del valor de x
guiente figura muestra la
dos de cerrojo: compartido,

	increment
o	conflicto
o	conflicto
o	compatible

nes ponen cerrojos en el
ccesan, el protocolo de dos
mite mayor concurrencia.



10

Mecanismos de Seguridad e Integridad

«¿Y qué era Abulafia, con su reserva secreta de files? Era el arca de lo que Belbo sabía, o creía saber, su Sophia. El elige un nombre secreto para penetrar en la profundidad de Abulafia, el objeto con que hace el amor (el único), pero mientras lo hace piensa en Lorenza, busca una palabra que conquiste a Abulafia y que al mismo tiempo también le sirva de talismán para poseer a Lorenza, quisiera penetrar en el corazón de Lorenza y comprender, así como puede penetrar en el corazón de Abulafia, quiere que Abulafia sea impenetrable para todos los demás, tan impenetrable como Lorenza lo es para él, se engaña pensando que custodia, conoce y conquista el secreto de Lorenza así como posee el de Abulafia.»

Umberto Eco (1932 -),
escritor italiano.

10.1 INTRODUCCION

Otras dos funciones importantes que deben ser soportadas por un SABD son las referentes a la seguridad y la integridad de los diferentes componentes de la base de datos.

La *seguridad* en un ambiente de bases de datos se refiere a la protección que debe tener una base de datos de cualquier usuario que no se encuentre autorizado a accesarla, mientras que la *integridad* se refiere a la protección de la base de datos de los usuarios que sí están autorizados en su acceso.

Así por ejemplo, si se considera una base de datos que administra los tratamientos que se le dan a un paciente en un hospital, deben considerarse estos dos conceptos. Por una parte, no todos tienen derecho a accesar los datos que conciernen a las enfermedades de los pacientes y por otra parte, se debe controlar que a cada paciente se le brinde el tratamiento que requiere.

Así, el SABD debe brindar mecanismos que garanticen la seguridad e integridad de los datos almacenados.

En el presente capítulo se estudian las diferentes técnicas de seguridad utilizadas en un ambiente de bases de datos. Posteriormente se definen los diferentes tipos de reglas que permiten mantener la integridad de la base de datos.

Los mecanismos que garantizan la seguridad en un sistema informático se pueden agrupar en cuatro grandes técnicas [DENN79]:

1. *El control del acceso a los datos.* En este caso el SABD verifica los derechos de acceso a los datos que tienen los usuarios de la base de datos.
- 2 *La seguridad multi-nivel.* Lo que se persigue con este tipo de técnicas es resguardar los caminos que toman los datos con el fin de evitar que lleguen a las manos de personas no autorizadas.

- 3 *El control de la inferencia.* Evitan que un usuario deduzca, a partir de un conjunto de datos a los cuales tiene acceso, información que no debe conocer.
- 4 *La criptografía.* Consiste en almacenar o transportar la información en forma tal que solo los usuarios que posean el código sean los que puedan comprenderla.

10.2 CONTROL DEL ACCESO A LOS DATOS

En una base de datos relacional, no existe una gran diferencia entre el administrador de la base de datos y un usuario que desea accesar dicha bases de datos.

Como una base de datos evoluciona en forma dinámica, el SABD debe brindar mecanismos *dinámicos de autorización*, capaces de tomar en cuenta, en cualquier momento, nuevos derechos de acceso o la supresión de otros.

Un usuario de un SABD puede crear relaciones, manipularlas como quiera y, si lo desea, autorizar a otros usuarios que puedan leer y/o modificar las relaciones. Por su parte, si un usuario desea suprimir una relación, el SABD debe anular automáticamente los derechos que eventualmente este usuario dio a otros sobre esta relación [DELO82].

Los mecanismos de autorización se aplican sobre las tablas en el sentido más amplio: a saber, tablas de base y vistas.

El SABD reconoce a todo usuario de una base de datos por una clave -del inglés *password*-.

En los catálogos de la base, esta clave sirve de prefijo a todos los nombres de objetos creados por este usuario.

Así, diferentes usuarios pueden tener objetos de igual nombre, como se aprecia en la figura 10.1.

De esta forma, por medio de la autorización, el SABD verifica que el usuario que accesa un objeto realmente tiene derecho a hacerlo.

de un usuario deduzca, a partir de lo que tiene acceso, información que

almacenar o transportar la información que los usuarios que posean el conocimiento.

S DATOS

entre una gran diferencia entre el usuario que desea accesar dicha

en forma dinámica, el SABD autorización, capaces de tomar los derechos de acceso o la

relaciones, manipularlas como los usuarios que puedan leer y/o un usuario desea suprimir una cláiticamente los derechos que tiene sobre esta relación [DELO82].

aplican sobre las tablas en el caso y vistas.

na base de datos por una clave

que sirve de prefijo a todos los

objetos de igual nombre, como

ación, el SABD verifica que el tiene derecho a hacerlo.

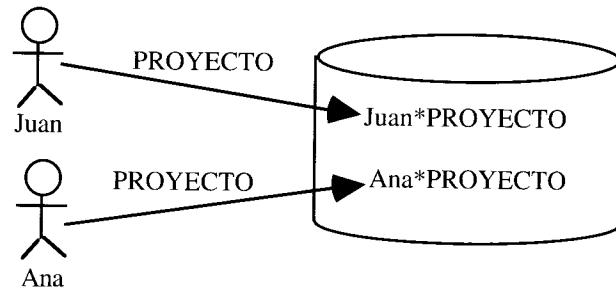


Figura 10.1 Ejemplo de objetos diferentes de la base de datos

10.2.1 Cláusula grant

Una vez que una persona ha creado un objeto, se convierte en su propietario y esto le da derecho sobre el mismo. Sin embargo, un objeto podría ser requerido por otros usuarios. Así, el lenguaje de consultas debería brindar la posibilidad de que algunos usuarios cedan derechos a otros usuarios sobre los objetos que les pertenecen.

Así, por ejemplo, el lenguaje relacional SQL ofrece dos cláusulas para autorizar y revocar derechos de acceso y que son el **grant** - autorización - y **revoke** - revocatoria -.

Los derechos sobre las relaciones, vistas o programas de aplicación se encuentran almacenados en el catálogo de la base de datos y se pueden consultar por medio del lenguaje.

Con respecto a los problemas de asignación de recursos, se debe hablar de dos categorías de usuarios. Por un lado, se tiene el dueño del recurso que se llama el *emisor* y por otra parte el usuario que recibe el privilegio. En este caso se habla del *receptor*. En la figura 10.2 se muestra tal situación.

La forma general de una cláusula **grant**, utilizando la sintaxis del SQLBase de Gupta, se presenta a continuación:

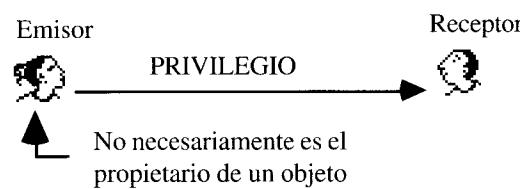


Figura 10.2 Emisor y receptor de un privilegio [DELO82]

```
grant privilegios
on nombre de la relación o de la vista
to lista de usuarios [public];
```

Cuando se utiliza la cláusula **public** significa que los privilegios son de orden público, es decir, se transmiten a cualquier usuario del sistema.

Los privilegios sobre una relación o vista son los siguientes:

- **select** Selecciona datos en una relación o vista.
- **insert** Introduce tuplas en una relación o vista
- **delete** Suprime tuplas de una relación o vista
- **update** Actualiza una relación y, opcionalmente, actualiza solo las columnas especificadas en la forma
 (col 1, col 2,..., col n)
- **index** Crea o suprime índices de una relación
- **alter** Altera una relación
- **all** Ejerce todos los privilegios anteriores

Ejemplo. Suponer que Ana fue quien creó la tabla EMPLEADO. Así, Ana es la propietaria de esta tabla. Ella quiere autorizar a Juan para que lea e introduzca tuplas en EMPLEADO:

Ana: **sql> grant insert, update on empleado to juan**

Receptor



privilegio [DELO82]

o de la vista
c]:

fica que los privilegios son de
quier usuario del sistema.

ta son los siguientes:

ación o vista.

ción o vista

ión o vista

pcionadamente, actualiza solo
en la forma

una relación

anteriores

tabla EMPLEADO. Así, Ana es la
a Juan para que lea e introduzca

juan

Esto significa que el usuario Juan puede transmitir los privilegios de inserción y modificación en la relación EMPLEADO.

La autorización de privilegios puede representarse por medio de un grafo dirigido, en donde los nodos representan a los usuarios involucrados y la dirección de los arcos va desde el emisor hacia el receptor. Así, la autorización del ejemplo anterior se puede representar según la figura 10.3.

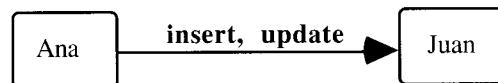


Figura 10.3 Ejemplo de grafo de autorización de privilegios

También, un usuario puede recibir privilegios de varios emisores, esto evita centralizar todos los poderes en manos de un solo individuo.

Ejemplo. Suponer que, a su vez, Juan transmite ciertos privilegios a Carlos y Mario y que Ana también les transmite en forma independiente.

Ana: `sql> grant insert, update, on empleado to juan`
 Juan: `sql> grant insert, update on empleado to carlos`
 Ana: `sql> grant insert on empleado to carlos`
 Ana: `sql> grant insert, index on empleado to mario`

Entonces el diagrama resultante aparece en la figura 10.4.

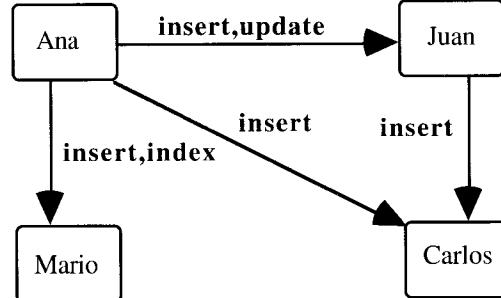


Figura 10.4 Ejemplo de autorización de privilegios

10.2.2 Cláusula revoke

Según lo que se mencionó anteriormente, la autorización de los privilegios puede generar un grafo dirigido bastante complejo. Ahora, ¿qué ocurre si algún emisor requiere quitarle los privilegios a alguno de sus receptores? Para ello existe una cláusula de revocatoria de privilegios, cuya sintaxis, usando el lenguaje SQL, se presenta a continuación:

```
revoke privilegios  

on nombre de la relación o vista  

from lista de usuarios [public];
```

Los privilegios que se pueden revocar son los mismos que se utilizan en la cláusula **grant**.

Ejemplo. Siguiendo el diagrama del ejemplo anterior, suponer que Ana decide quitarle los privilegios a Juan en la tabla EMPLEADO

Ana: **sql> revoke read, update on empleado from Juan**

Sin embargo, el problema de la revocatoria no es tan directo como aparece en la figura 10.5.

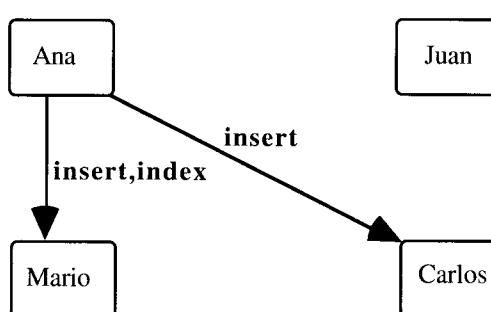


Figura 10.5 Ejemplo de revocatoria de privilegios

En efecto, considere el siguiente ejemplo.

Ejemplo. Sea la figura 10.6. En este caso se están representando dos trayectorias diferentes de asignación de recursos. Ahora, ¿qué ocurriría si, por ejemplo, Juan decide quitarle el privilegio a Carlos? Carlos brindó el privilegio a Cecilia y Pedro, sin embargo, solo debe desaparecer el privilegio que le ha brindado a Cecilia y no a Pedro, pues esta asignación se hizo por medio de Mario y él le mantiene el privilegio a Carlos.

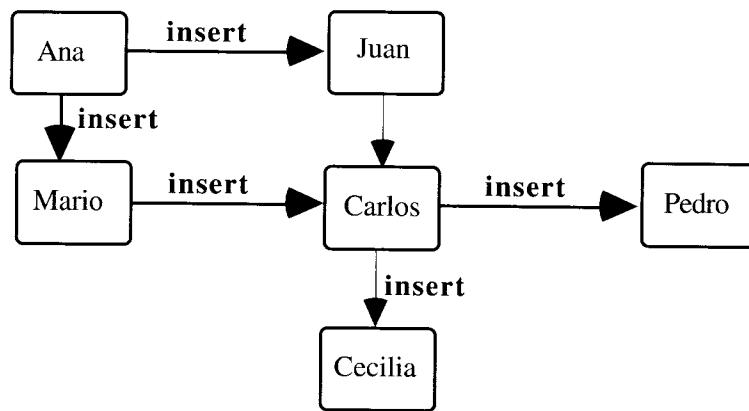


Figura 10.6 Ejemplo de revocatoria de privilegios

Como se puede ver la revocatoria de privilegios no es un problema sencillo. Es por esta razón que a cada asignación que se haga de un recurso se le asigna una *estampilla de tiempo* y cada vez que se aplica la cláusula `revoke`, se aplica la siguiente regla:

Regla de revocatoria

Se revoca un privilegio acordado a un desautorizado si este privilegio se dio antes de cualquier otro privilegio que aun posea el desautorizado

Ejemplo. Sea el diagrama de la figura 10.7 que representa una asignación de un mismo recurso a varios usuarios.

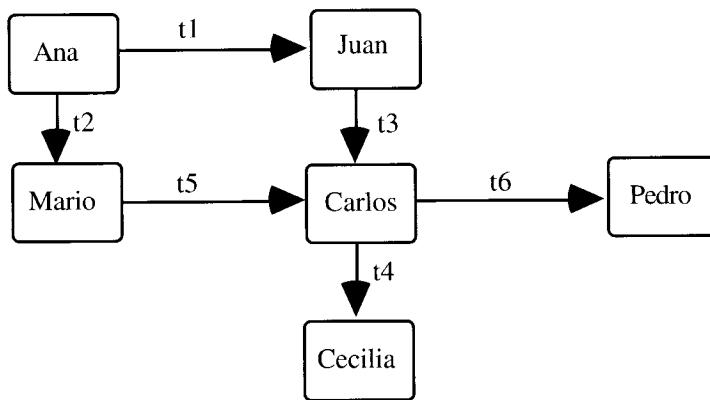


Figura 10.7 Ejemplo de asignación de un mismo recurso con estampillas de tiempo

Suponer que Juan revoca el privilegio dado a Carlos en el tiempo t3. En este caso se debe suprimir el privilegio que le brindó Carlos a Cecilia y dejar el privilegio que brindó Carlos a Pedro, como se aprecia en la figura 10.8.

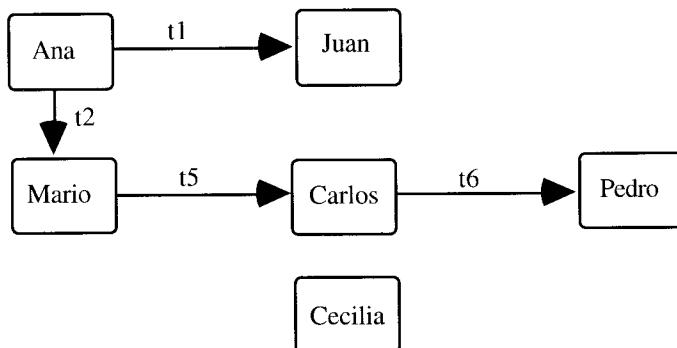
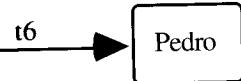


Figura 10.8 Revocatoria de un mismo privilegio, usando la regla de revocatoria

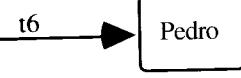
10.3 LA SEGURIDAD MULTI-NIVEL

El problema del control de los flujos de datos se ilustra en la figura 10.9.



curso con estampillas de tiempo

los en el tiempo t3. En este caso
los a Cecilia y dejar el privilegio
a figura 10.8.



usando la regla de revocatoria

tos se ilustra en la figura 10.9.

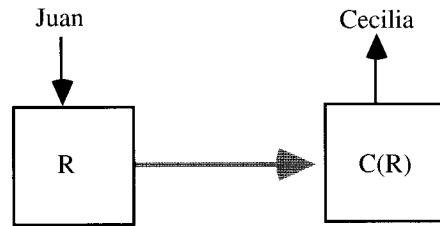


Figura 10.9 Control de flujos

Cecilia y Juan son usuarios de un SABD. Cecilia no tiene el derecho de acceso a la relación R, entonces, solicita a Juan, que sí tiene derecho, le brinde una copia C(R) de esta relación.

En general, se puede decir que un *flujo* existe entre un objeto X y un objeto Y cuando un programa lee X y hace escrituras sobre Y.

Un ejemplo clásico de un flujo es el copiar un archivo X en el archivo Y.

La mayoría de los métodos de control de flujos utilizan niveles de seguridad. Así, los objetos de la base de datos se dividen en clases y cada una de ellas cuenta con un grado de seguridad. Según [DENN79] la transferencia de información entre un emisor y un receptor es posible solo si el nivel de seguridad del receptor es al menos tan privilegiado que el del emisor.

El protocolo más sencillo de control de flujos comprende dos niveles: *libre* y *confidencial*.

En este caso se permite cualquier flujo de datos excepto aquellos que van del nivel confidencial al nivel libre, según se aprecia en la figura 10.10.

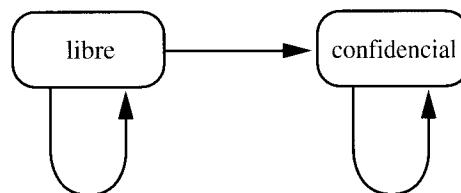


Figura 10.10 Autorización de flujos

Otros sistemas más complejos pueden requerir varios niveles de seguridad y por ende utilizar protocolos más complejos.

Así, cada nivel de seguridad se puede representar por medio de un par (*nivel de autorización, conjunto de categorías*)

Según [DENN79] se pueden establecer al menos cuatro niveles de autorización:

- *No clasificado*
- *Confidencial*
- *Secreto*
- *Muy secreto*

Para estos niveles de autorización se puede establecer el siguiente orden:
no clasificado < confidencial < secreto < muy secreto

Así, las categorías de objetos están formadas considerando combinaciones de características elementales sobre objetos. Por lo general se pueden considerar cuatro grandes categorías:

- *Libre*,
- *Restringido*,
- *Sensitivo*,
- *Codificado*

El protocolo de control de flujos en este caso se puede presentar de la siguiente forma:

Un flujo de información del nivel de seguridad (i,x) al nivel de seguridad (j,y) es posible si:

1. $i \leq j$
2. $x \subseteq y$

Ejemplo. Los siguientes flujos de datos son posibles:

[Se-]
[Se-]
Los
[Se-]
[Se-]

10.4 C

Otr
de dat
una pe
a parti

Par
[DEN]

Eje

Preg
• Se
• Ec
• Ca
• Do
• Gr
• Ge

Resp
quier
confi

Preg
• Se
• Ed
• Ca
• Do
• Gr
• Ge
• To
Resp

querir varios niveles de complejos.

entar por medio de un par
categorías)

menos cuatro niveles de

ablecer el siguiente orden:

Muy secreto

formadas considerando
s sobre objetos. Por lo
categorías:

o se puede presentar de la

guridad (i,x) al nivel de

s:

[Secreto | Restringido, Sensitivo]→ [Muy secreto | Restringido, Sensitivo],
[Secreto | Sensitivo]→ [Secreto | Restringido, Sensitivo, Codificado].

Los siguientes flujos de datos no son posibles:

[Secreto | Restringido, Sensitivo]→[Libre | Restringido, Sensitivo]
[Secreto | Restringido, Sensitivo]→[Secreto | Restringido]

10.4 CONTROL DE LA INFERENCIA

Otra de las técnicas utilizadas para mantener la seguridad de una base de datos es por medio del control de la inferencia, es decir, evitando que una persona pueda, sin tener autorización, conocer e inferir información a partir de los datos a cuales sí tiene acceso.

Para entender mejor esto se va a introducir el siguiente ejemplo [DENN79].

Ejemplo. Control sobre el tamaño del resultado:

Pregunta: ¿Cuántos enfermos hay con las siguientes características?

- Sexo masculino
- Edad 45-50
- Casado
- Dos niños
- Graduado del ITCR
- Gerente de la Empresa AREX

Respuesta: Suponer que la persona que hace esta pregunta sabe que es Arguedas quien posee estas características. Ahora, desea descubrir informaciones confidenciales sobre Arguedas, por ejemplo.

Pregunta: ¿Cuántos enfermos hay con las siguientes características?

- Sexo masculino
- Edad 45-50
- Casado
- Dos niños
- Graduado del ITCR
- Gerente de la Empresa AREX
- Toma calmantes debido a depresiones

Respuesta: La respuesta será 1 si Arguedas toma calmantes, 0 en caso contrario.

Así, la persona no autorizada encuentra una condición C en donde el conjunto resultante solo tiene un elemento.

Por lo tanto, ¿cuántos individuos satisfacen C y X? Si la respuesta es 1 el individuo posee la característica X, 0 en caso contrario.

Así, la regla que debe respetarse es la siguiente [HOFF70]:

No responder a las consultas en donde hay menos de k o más de n-k tuplas en la respuesta.

En este caso n es el número total de tuplas de la base de datos y k es un entero positivo que especifica el tamaño mínimo permitido de la consulta.

Si el lenguaje de consultas permite el complemento, un tamaño máximo de n-k debe también controlarse, pues de lo contrario pueden hacer preguntas de tipo NOT C.

Sin embargo, este tipo de control no es del todo satisfactorio.

En efecto, cuando k es aproximadamente igual a n/2, las fugas son posibles usando la técnica del *perseguidor* [DENN79].

La idea de base es suponer que a un *indiscreto* se le informó que una persona P está caracterizada por la condición

$$C = A \text{ AND } B.$$

Además, se supone que el SABD puede responder a las preguntas caracterizadas por la condición A y las caracterizadas por la condición A AND NOT B. La condición

$$T = A \text{ AND NOT } B$$

se llama el *perseguidor* de P.

Este concepto fue introducido por Schlörer [SCHL75] y se llama perseguidor pues quien hace las preguntas puede usarlo para localizar características adicionales de un individuo.

una condición C en donde el
en C y X? Si la respuesta es
n caso contrario.

guiente [HOFF70]:

ay menos de k o más de n-k

das de la base de datos y k es
ño mínimo permitido de la

el complemento, un tamaño
pues de lo contrario pueden

del todo satisfactorio.

nte igual a n/2, las fugas son
[DENN79].

discreto se le informó que una
ón

3.

de responder a las preguntas
acterizadas por la condición A

OT B

chlörer [SCHL75] y se llama
as puede usarlo para localizar
o.

Ejemplo. [DENN79] Sea la contribución de varias personalidades a un cierto partido político y que se representa en la relación de la figura 10.11. Por otra parte, suponer que se pueden hacer consultas cuyas respuestas involucran solo sumas y/o cardinalidades de conjuntos.

CONTRIBUCION

Nombre	Sexo	Profesión	Monto
Alonso	m	Periodista	30000
Bernardo	m	Periodista	5000
Carlos	m	Empresario	1000
Sara	f	Periodista	50000
Sonia	f	Profesora	1000
Tomás	m	Profesor	20000
Uriel	m	Médico	200000
Verónica	f	Abogada	100000

Figura 10.11 Ejemplo de relación

Suponer que C = (periodista AND femenino) identifica en forma única a Sara. Con el perseguidor T= (periodista AND NOT femenino) = (periodista AND masculino), se conocerá la contribución de Sara:

P: ¿Cuántas personas son periodistas?

R: 3

P: ¿Cuántas personas son periodistas y hombres?

R: 2

Así C identifica a Sara.

P: ¿Cuál es la contribución total de los periodistas?

R: 85000

P: ¿Cuál es la contribución total de los periodistas varones?

R: 35000

Por lo tanto, Sara contribuyó con 50000 colones.

10.5 LA CRIPTOGRAFIA

Tanto para el caso de los controles de flujo, de acceso o de inferencia no se puede evitar que un operador deje en su "mesa de trabajo" una lista de *passwords* de usuarios o que un indiscreto vaya

a consultar las informaciones que pueden estar en algún dispositivo de respaldo.

Para evitar este tipo de indiscreción se puede cifrar la información de tal manera que solo pueda ser descifrada por personas -individuos o programas- que conozcan el código secreto.

Este método de cifrar mensajes se conoce como *criptografía* y es muy seguro para el almacenamiento y transporte de la información.

A continuación se van a ver los principales elementos de un sistema criptográfico.

En primer lugar, se tiene lo que se llama el *código* que es un método secreto de escritura, en donde el mensaje que se desea almacenar o transportar en una red es codificado y se llama *criptograma*. El proceso de transformar un mensaje en uno codificado se denomina *encriptamiento*. Por otra parte, el proceso inverso de transformar un criptograma en un mensaje se denomina *desencriptamiento*.

En la figura 10.12 se pueden apreciar estos conceptos.

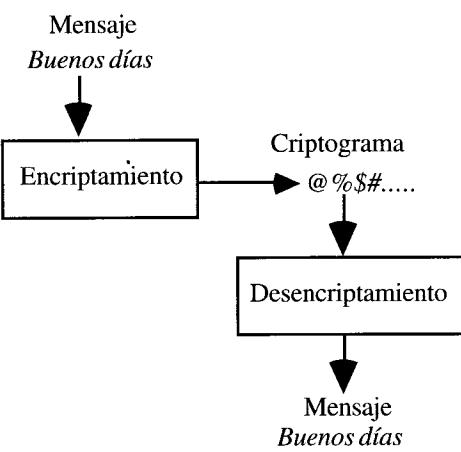


Figura 10.12 Proceso de encriptamiento de un mensaje

n estar en algún dispositivo

puede cifrar la información de
a por personas -individuos o
o.

e como *criptografía* y es muy
e de la información.

ales elementos de un sistema

llama el *código* que es un
el mensaje que se desea
es codificado y se llama
mar un mensaje en uno
Por otra parte, el proceso
en un mensaje se denomina

tos conceptos.



De una manera más formal, se puede decir que un sistema criptográfico consta de cinco componentes:

- Un mensaje M que debe almacenarse o transportarse.
- Un criptograma C
- Un conjunto de llaves I
- Un conjunto de reglas de encriptamiento
 $e_i : M \rightarrow C$ con $i \in I$
- Un conjunto de reglas de desencriptamiento
 $d_j : C \rightarrow M$ con $j \in I$

Como ejemplo de un algoritmo de encriptamiento, se puede mencionar el algoritmo de *cifrados de transposición*. Este tipo de algoritmo reordena los caracteres que componen el mensaje según un esquema establecido.

Así, por ejemplo, el algoritmo de *transposición columnar*, toma un mensaje y lo escribe en una matriz $m \times n$ y el criptograma es el que se genera leyendo el mensaje en forma columnar.

Ejemplo. Considerar el mensaje: *ser o no ser, he ahí la cuestión* y se desea utilizar una matriz de $n \times 5$. En este caso, dicha oración queda ubicada de la siguiente forma

s	e	r		o
n	o			s
e	r	,		h
e		a	h	í
	l	a		c
u	e	s	t	i
ó	n			

Así el criptograma sería el siguiente:

s ee uó enr len ro,aas h t oshíci.

Por supuesto que dicho algoritmo es muy sencillo y es relativamente fácil de descifrar.

Sin embargo, existen otros tipos de algoritmos más complejos y que permiten alta confiabilidad. Este es el caso de los métodos de encriptamiento llamados de *clave pública*, inventados en 1976 por W. Diffie y M. Hellmann de la Universidad de Stanford. En este caso el código de encriptamiento no puede invertirse fácilmente para hallar el código de desencriptamiento.

El sistema de encriptamiento público más conocido es el llamado RSA, inventado en 1978 por R. Rivest, A. Shamir y L. Adleman del Massachusetts Institute of Technology. El método se basa en las dos suposiciones siguientes [COHE95]:

- Es relativamente fácil determinar dos números primos grandes p y q y multiplicarlos para obtener un número $n = p \cdot q$ que sirve de base para un código de encriptamiento.
- El proceso inverso, es decir, determinar los números primos p y q a partir del número n es difícil realizarlo en un tiempo aceptable cuando n es un número muy grande.

Ejemplo [COHE95]. María desea enviar un mensaje a Carlos, quien escoge dos números primos p y q cuya división por 3 tenga a 2 como resto. Así, calcula $n = p \cdot q$ y se lo envía a María. Por su parte, María transforma el mensaje en una sucesión de dígitos -lo cual resulta muy fácil- y la divide en segmentos de longitud n . Sea uno de esos segmentos representado por el número x . María procede a calcular el resto y al dividir x^3 por n el segmento codificado es el número y . Cuando Carlos recibe el mensaje utiliza un número ultrasecreto $e = (2(p-1)(q-1) + 1)/3$ el cual es entero por las escogencias de los números primos p y q . El fragmento original por su parte, es decir el número x , es igual al resto de dividir y^e por n . En este caso a una persona le será imposible descifrar en un tiempo oportuno el criptograma, aunque conozca el número n .

Así, aunque se conozca el número n , se requiere factorizar $p \cdot q$ con el fin de descifrar el mensaje. Sin embargo, si los números primos escogidos son muy grandes -de varios cientos de dígitos-, se requeriría

mucho ti
para logr

Actua
informac

10.6 REC

En el
como la
referencia
utilizando

Existe
considera
operacion

Las op
insertar, s
efectos de
misma rela

Ejemplo.

- Un c
- Una e
- Las u

Las ope
involucrar:

- Varios
- Dos o
- Un atr

y sencillo y es relativamente

rítmos más complejos y que
caso de los métodos de
inventados en 1976 por W.
le Stanford. En este caso el
rse fácilmente para hallar el

nás conocido es el llamado
.. Shamir y L. Adleman del
método se basa en las dos

números primos grandes p y
ero $n = p \cdot q$ que sirve de base

ar los números primos p y q
arlo en un tiempo aceptable

nsaje a Carlos, quien escoge dos
2 como resto. Así, calcula $n = p \cdot q$
orma el mensaje en una sucesión
en segmentos de longitud n . Sea
ero x . María procede a calcular el
do es el número y .
ero ultrasecreto $e = (2(p-1)(q-1))$
los números primos p y q . El
ro x , es igual al resto de dividir y^e
e descifrar en un tiempo oportuno

equiere factorizar $p \cdot q$ con el
o, si los números primos
tos de dígitos-, se requeriría

mucho tiempo de máquina -inclusive de los computadores más rápidos- para lograr tal factorización.

Actualmente, se considera que la técnica para seguridad de la información más barata y segura es la criptografía.

10.6 REGLAS DE INTEGRIDAD

En el capítulo 3 se estudiaron varios tipos de reglas de integridad, como la integridad de relación, integridad de dominio e integridad referencial y se vio cómo estas reglas de integridad pueden programarse utilizando el SQL.

Existe otro tipo general de reglas de integridad y que podrían considerarse como generalización de las anteriores y que se denominan *operaciones de disparo* - del inglés *triggers*-.

Las operaciones de disparo son reglas que regulan la validez de insertar, suprimir, actualizar y recuperar operaciones, incluyendo los efectos de éstas sobre otras relaciones u otros atributos dentro de la misma relación.

Ejemplo. Requerimientos que generan operaciones de disparo:

- Un cliente no puede tener un crédito superior a € 4000000
- Una empresa no puede exceder el límite de crédito establecido para cada uno de sus clientes
- Las unidades solicitadas en una orden de cliente deben ser múltiplos del número de unidades por paquete del producto

Las operaciones de disparo reflejan tipos de restricciones que pueden involucrar:

- Varios atributos de diferentes relaciones
- Dos o más atributos en una misma relación
- Un atributo o relación y un parámetro externo

Las operaciones de disparo tienen dos componentes lógicos. Por una parte, se tiene el *disparador*, o evento y condición que causa que la operación suceda y la *operación*, o la acción de disparo.

Cuando se establece una operación de disparo es recomendable considerar al menos los siguientes componentes [FLEM89]:

- Evento que provoca la operación de disparo -inserción, actualización, supresión o recuperación-
- Objeto del evento -nombre de la relación y/o atributo que va a ser modificado o accesado-
- Condición bajo la cual se inicia la operación de disparo
- Acción que se lleva a cabo -tal como evento rechazado o evento asociado al disparador-

Ejemplo. La operación de disparo generada por el primer requerimiento del ejemplo anterior puede representarse en un diccionario de datos de la siguiente forma:

Requerimiento	Evento	Relación	Atributo	Condición	Acción
Un cliente no puede tener un crédito superior a ¢ 4000000	Inserción	CLIENTE	Crédito	Si Monto es superior a ¢4000000	Rechazar la inserción

EJERCICIOS Y PREGUNTAS DE REPASO

- 10.1 Clasificar los diferentes tipos de control de seguridad que se pueden tener en un sistema informático.
- 10.2 Establecer cuatro diferentes niveles de seguridad con 4 categorías de objetos. Enunciar el protocolo de control de flujos y establecer, bajo estas premisas, varios flujos de control válidos.
- 10.3 Considerar el diagrama de asignación de un mismo recurso que se muestra en la figura 10.13. Suponga que Mario decide quitarle el

10.4 Ut

par

L
pue10.5 ¿Qu
la ir

componentes lógicos. Por una condición que causa que la operación de disparo.

La operación de disparo es recomendable para los siguientes factores [FLEM89]:

- Operación de disparo -inserción, actualización-

- Operación y/o atributo que va a ser actualizado

- Operación de disparo

- Evento rechazado o evento de error

por el primer requerimiento del operador de datos de la siguiente forma:

Atributo	Condición	Acción
Término	Si Monto es superior a \$4000000	Rechazar la inserción

SO

control de seguridad que se aplica.

de seguridad con 4 categorías de control de flujos y establecer. control válidos.

ón de un mismo recurso que se aplica que Mario decide quitarle el

privilegio al usuario Carlos en el tiempo t4. Aplique la regla de revocatoria para establecer qué asignaciones quedan.

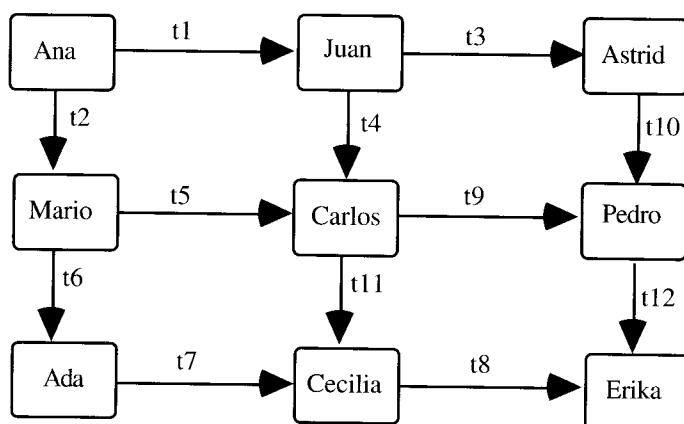


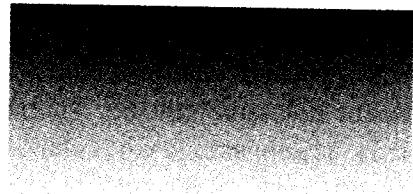
Figura 10.13 Ejemplo de una asignación de privilegios

- 10.4 Utilizar el algoritmo de transposición columnar con una matriz $n \times 7$ para encriptar el siguiente mensaje:

La imprenta es un ejército de 26 soldados de plomo con el que se puede conquistar el mundo.

J. Gutenberg

- 10.5 ¿Qué es una operación de disparo y cómo puede ayudar a mantener la integridad de las bases de datos?

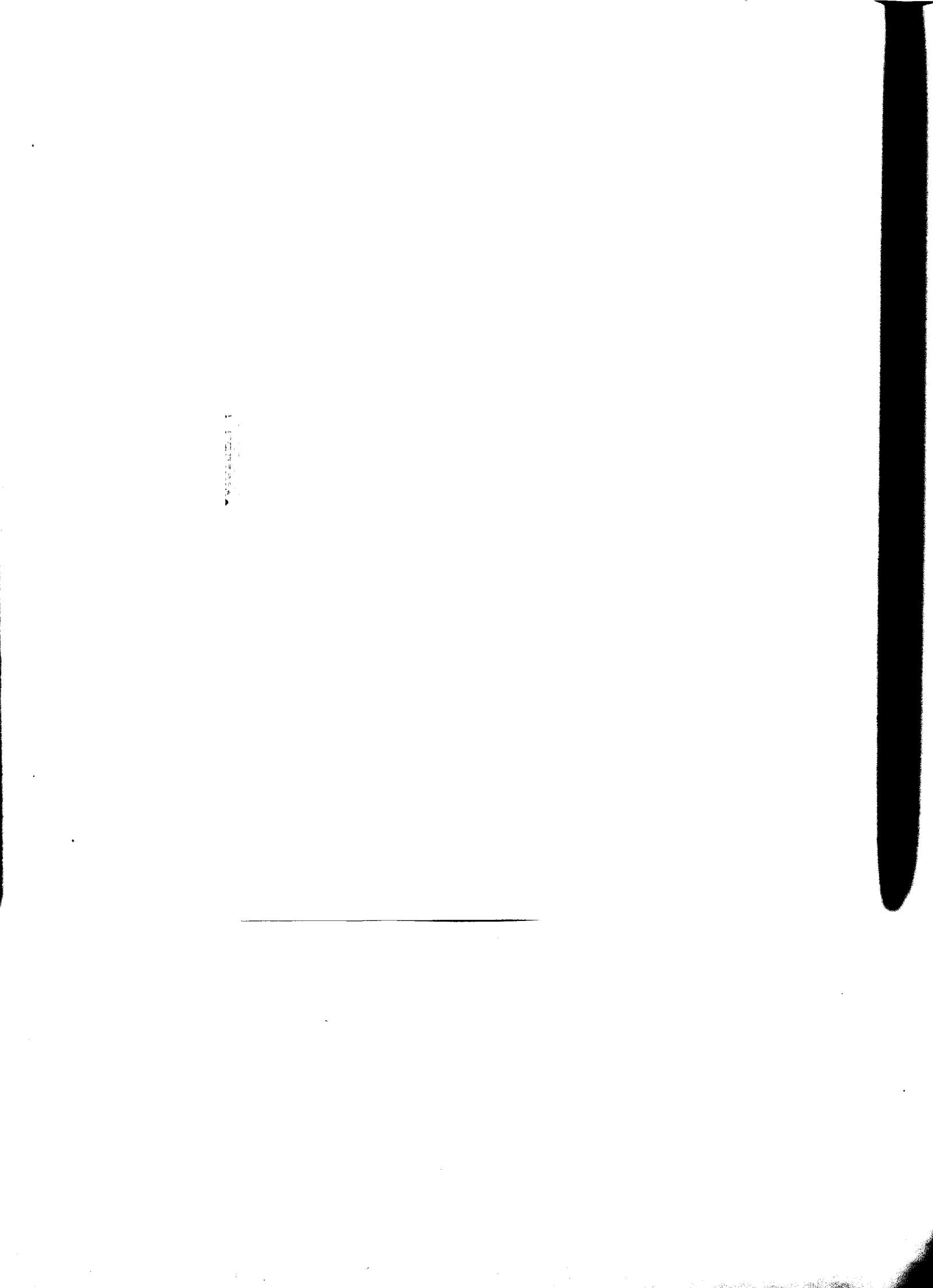


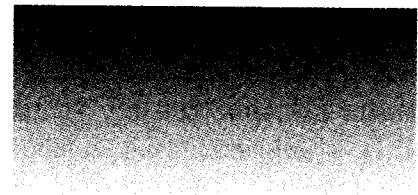
11

Recuperación Después de Fallas

«Es imposible volver a la infancia; nadie puede hacerlo. Es imposible regalar sus cañones, sus máquinas y su dinero y establecerse en una pequeña y pacífica ciudad para escribir poesías y tocar sonatas. Pero es posible recorrer el camino que también ha de seguir el individuo cuando su vida le ha llevado a errores y profundos sufrimientos. Es posible recordar el camino anterior, el propio origen y la propia infancia, el desarrollo, el apogeo y el fracaso, y encontrar así en el camino de estos recuerdos las fuerzas que pertenecen a cada uno esencial e irreversiblemente. Es preciso “mirar hacia dentro”, como dicen los piadosos. Y en lo más íntimo de nuestro ser encontraremos fácilmente la propia identidad, y esta identidad no querrá escapar a su destino, sino que lo aceptará y empezará de nuevo con la integridad recuperada.»

Hermann Hesse (1877-1962),
poeta y escritor alemán





11

Recuperación Después de Fallas

«Es imposible volver a la infancia; nadie puede hacerlo. Es imposible regalar sus cañones, sus máquinas y su dinero y establecerse en una pequeña y pacífica ciudad para escribir poesías y tocar sonatas. Pero es posible recorrer el camino que también ha de seguir el individuo cuando su vida le ha llevado a errores y profundos sufrimientos. Es posible recordar el camino anterior, el propio origen y la propia infancia, el desarrollo, el apogeo y el fracaso, y encontrar así en el camino de estos recuerdos las fuerzas que pertenecen a cada uno esencial e irreversiblemente. Es preciso “mirar hacia dentro”, como dicen los piadosos. Y en lo más íntimo de nuestro ser encontraremos fácilmente la propia identidad, y esta identidad no querrá escapar a su destino, sino que lo aceptará y empezará de nuevo con la integridad recuperada.»

Hermann Hesse (1877-1962),
poeta y escritor alemán

11.1 INTRODUCCION

Un sistema de bases de datos reside en la memoria secundaria que por lo general se compone de discos magnéticos. En una situación ideal, cuando un usuario utiliza la base de datos para consultarla y/o modificarla, el SABD debe enviar los datos involucrados a la memoria principal y cuando se hicieron las modificaciones, éstas deben ser enviadas asimismo a la memoria secundaria. Sin embargo, cuando se realizan las transacciones se pueden presentar problemas debido a fallas. De ahí, que el SABD debe brindar una serie de mecanismos que permitan recuperar una base de datos -es decir, dejarla en un estado consistente- si una de tales fallas ocurre. Las fallas que pueden presentarse en el momento más inoportuno se pueden clasificar en las siguientes:

- *Fallas de una transacción.* Este tipo de fallas se da cuando una transacción no puede llegar a su punto de validación debido a un error del programa. También, estos problemas pueden ocurrir cuando se presenta un interbloqueo entre transacciones, o cuando se ejecuta en forma indebida algún mandato del SABD.
- *Fallas del sistema.* Estas fallas requieren que el sistema sea detenido para arrancarlo nuevamente. En este caso solo se pierden los datos que se encontraban en la memoria principal -que es una memoria volátil-.
- *Fallas de la memoria secundaria.* Una falla en los discos se puede presentar debido a un desperfecto del *hardware* -cabezas sucias, errores de paridad, etc.-, o problemas con el *software* que pueden provocar escrituras incorrectas, como por ejemplo un transferencia incorrecta de los datos. Este tipo de problemas tienen como consecuencia la pérdida parcial de la base de datos.

Según varios estudios, se considera que el 97 por ciento de las transacciones se realizan en forma satisfactoria. Del porcentaje restante, el 2 por ciento se cancela debido a los usuarios, ya sea por inserciones

incorrectas de las transacciones o por cancelaciones. El 1 por ciento restante se debe al sistema cuando encuentra algún tipo de interbloqueo y debe abortar la transacción.

En la figura 11.1 se presentan las frecuencias y los tiempos de recuperación de estas fallas, según se presenta en [GRAY81].

Tipo de falla	Frecuencia	Tiempo de recuperación
Transacción	Varios por minuto	Milisegundos
Sistema	Varios por mes	Segundos
Discos	Varios por año	Minutos

Figura 11.1 Frecuencia y tiempo de recuperación de las fallas

Así, es importante analizar las técnicas que permiten al SABD recuperar la base de datos. Se debe mencionar que existen otros tipos de fallas que pueden provocar una pérdida parcial o total de la base de datos en forma irrecuperable, como puede ser una explosión, incendio o inundación. De ahí la importancia de contar con copias de la base de datos en lugares seguros.

De las técnicas de recuperación se estudiarán en las próximas secciones las llamadas técnicas de paginación y de la bitácora. Previamente se estudia el problema de las fallas cuando se realizan varias transacciones en forma concurrente.

11.2 TRANSACCIONES CONCURRENTES

Suponer un sistema en que se están realizando 5 transacciones como se muestra en la figura 11.2.

En el instante t_i se produce una falla. Como se aprecia en la figura 11.1, las transacciones T_2 , T_4 y T_5 han llegado a su punto de validación antes de que haya ocurrido la falla. Debido a esto, el SABD está en la obligación de restaurar y validar las modificaciones de los datos provocados por estas transacciones.

cancelaciones. El 1 por ciento
para algún tipo de interbloqueo

ecuencias y los tiempos de
nta en [GRAY81].

Tiempo de recuperación

Milisegundos

Segundos

Minutos

operación de las fallas

que permiten al SABD recuperar
existen otros tipos de fallas que
de la base de datos en forma
incendio o inundación. De ahí
de datos en lugares seguros.
estudiarán en las próximas
ginación y de la bitácora.
as fallas cuando se realizan

ES
lizando 5 transacciones como

Como se aprecia en la figura
gido a su punto de validación
o a esto, el SABD está en la
modificaciones de los datos

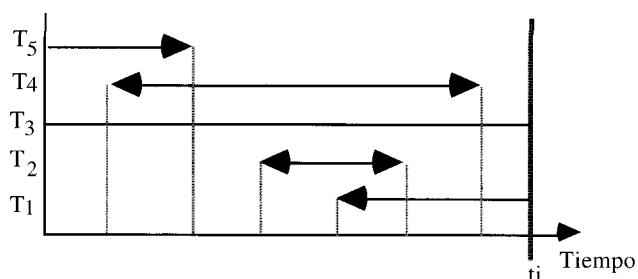


Fig. 11.2 Ejemplo de falla en transacciones concurrentes

Por otra parte, las transacciones T_1 y T_3 no alcanzaron su punto de validación. Esto significa que el SABD debe *deshacer* el recorrido de las mismas como si nunca se hubieran realizado.

A continuación, se introduce la técnica llamada de paginación.

11.3 TECNICA DE PAGINACION

La técnica de paginación presentada inicialmente en [LORI77] se basa en un mapeo dual entre las páginas y sus respectivas ubicaciones en los discos. En este contexto, un mapeo representa el estado actual de un segmento que ha sido modificado y el otro representa el estado inmediatamente anterior. De esta forma se garantiza que siempre se puede restaurar la base de datos en un estado consistente.

11.3.1 Concepto de página

Una base de datos se almacena físicamente en un conjunto de espacios llamados *segmentos* S_i , los cuales se componen de páginas, las que contienen las tuplas que contienen las relaciones, según se aprecia en la figura 11.3.

La *página* es la unidad de asignación física en la memoria secundaria y además es la unidad de transferencia entre la memoria secundaria y la memoria principal.

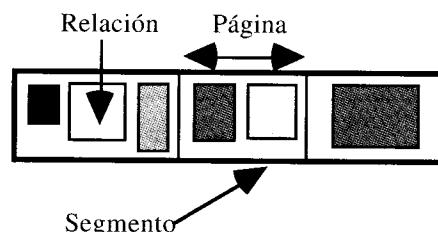


Figura 11.3 Segmentos de una base de datos

Para cada segmento, la asignación de las páginas se hace en forma dinámica. Si la base de datos crece más de lo previsto, se puede ampliar el espacio real.

Se proveen dos funciones que permiten manipular los segmentos. En primer lugar, se tiene una operación de *apertura del segmento* que permite el acceso al segmento con el fin de manipular su contenido. Y en segundo lugar se tiene la *cerradura del segmento* que asegura que el contenido del segmento se encuentra protegido.

11.3.2 Tabla de páginas y vector de ocupación de bloques físicos

Debido a la asignación dinámica de las páginas en los discos, cada segmento tiene asociado lo que se denomina *tabla de páginas* [HAER83].

La tabla de páginas del segmento, denotado por TP, es un vector de n campos, en donde n representa el número de páginas del segmento. Este vector apunta a las páginas de datos. De esta forma, la notación

$$TP[i] = j$$

significa que la página i del segmento se encuentra ubicado en la página j de datos del disco.

Si $TP[i] = @$ -valor nulo-, significa que la página i en el disco está disponible. Además, cada segmento tiene asociado un *vector de ocupación*

denotado por VO de tal manera que si $VO[i] = 1$ significa que la página i en disco está ocupada, en caso contrario se tiene que $VO[i] = 0$.

En la figura 11.4 se muestra un segmento con sus vectores asociados.

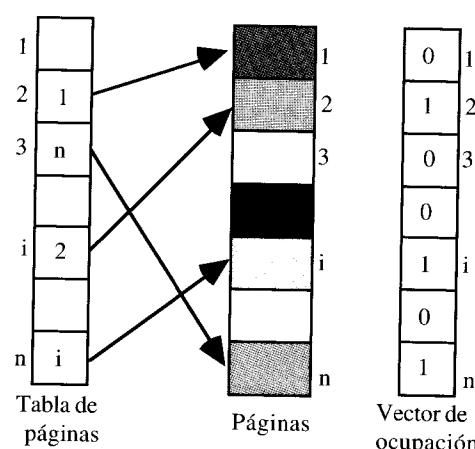


Figura 11.4 Ejemplo de asignación de páginas de un segmento

11.3.3 Transferencia de páginas

Con el fin de establecer la transferencia de páginas entre la memoria principal y la memoria secundaria, ésta última cuenta con pequeñas memorias conocidas como *buffers*. Una palabra en español para este podría ser el de *amortiguador*, sin embargo, el término de buffer se impuso y es universalmente utilizado.

Para la transferencia de la páginas se tienen dos buffers. El primero es el llamado buffer de *tablas de páginas* y que se denota por BTP. En esta memoria se ubica el vector de página del segmento en que se localiza la página que debe trasladarse.

El segundo se denomina simplemente el *buffer de páginas* y se denota por BP. En este buffer se ubicarán las páginas que fueron señaladas por el buffer BTP [DELO82].

denotado por VO de tal manera que si $VO[i] = 1$ significa que la página i en disco está ocupada, en caso contrario se tiene que $VO[i] = 0$.

En la figura 11.4 se muestra un segmento con sus vectores asociados.

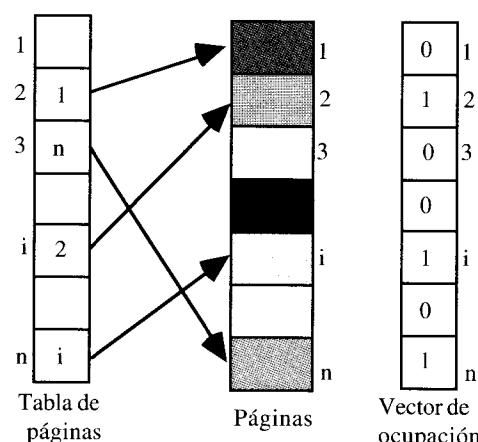


Figura 11.4 Ejemplo de asignación de páginas de un segmento

11.3.3 Transferencia de páginas

Con el fin de establecer la transferencia de páginas entre la memoria principal y la memoria secundaria, ésta última cuenta con pequeñas memorias conocidas como *buffers*. Una palabra en español para este podría ser el de *amortiguador*, sin embargo, el término de buffer se impuso y es universalmente utilizado.

Para la transferencia de la páginas se tienen dos buffers. El primero es el llamado buffer de *tablas de páginas* y que se denota por BTP. En esta memoria se ubica el vector de página del segmento en que se localiza la página que debe trasladarse.

El segundo se denomina simplemente el buffer de *páginas* y se denota por BP. En este buffer se ubicarán las páginas que fueron señaladas por el buffer BTP [DELO82].

Una vez que se tenga la página deseada en la memoria principal, ésta puede ser consultada y(o) modificada. Posteriormente, cuando se termina la modificación, la página es enviada a los discos pero no donde se encontraba anteriormente, sino a un nuevo espacio o página física en los discos.

En la figura 11.5 se representan las diferentes operaciones para enviar la página i de un segmento que se encuentra en la página de datos j.

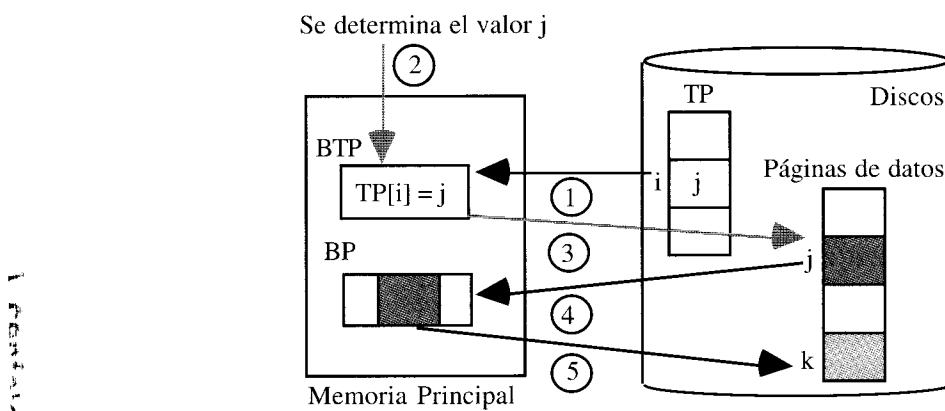


Figura 11.5 Operaciones para el intercambio de páginas

Estas operaciones son las siguientes:

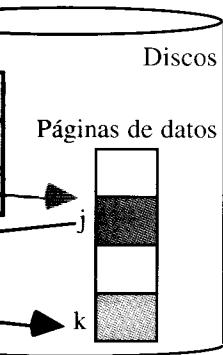
1. Se envía el vector TP al buffer de tablas de páginas BTP.
2. Se determina la ubicación de la página i, en este caso es j.
3. Se va a buscar la página de datos j.
4. Se envía al buffer de páginas BP la página de datos j.
5. La página modificada es enviada a una nueva página de datos k.

11.3.4 Recuperación de segmentos

Bajo la filosofía de transferencia de páginas entre la memoria principal y los discos, ¿qué pasa si ocurre una falla cuando se están llevando a cabo estas operaciones de modificación? Podría dejar la base

en la memoria principal, ésta se pierde permanentemente, cuando se termina la operación pero no donde se encontraba la página física en los discos.

Hay otras operaciones para enviar datos en la página de datos j.



Cambio de páginas

as de páginas BTP.

a i. en este caso es j.

Página de datos j.

Nueva página de datos k.

páginas entre la memoria y una falla cuando se están realizando operaciones? Podría dejar la base de datos inconsistente.

de datos en un estado inconsistente. Por esta razón, cuando se envía una página al buffer BP, se le asocia un *indicador de modificación* que servirá para indicar si la página fue modificada o no cuando estuvo en la memoria principal.

Cuando ocurre una falla y si la tabla de páginas TP no se recopió completamente en disco, la correspondencia entre páginas y sus posiciones podría perderse. Igualmente, si algunas páginas no se han recopilado, su contenido último no reflejará las últimas modificaciones hechas por el usuario. Se debe entonces ampliar este mecanismo para que se pueda recuperar el contenido de las páginas. Entonces se adicionan dos nuevas funciones. La primera es la *protección* de un segmento y la segunda la *restauración* de un segmento.

Además, el sistema no solo trabaja con un vector TP; cada segmento cuenta con una segunda tabla de páginas llamada *sombra* y denotada por TPS, que sirve para mantener el último estado del segmento antes de que se lleve a cabo la solicitud de la página. Así, los dos vectores de página asociados a cada segmento van a ser utilizados. En efecto, uno estaría reflejando el nuevo estado que va a modificar la base de datos y el otro vector se quedaría con el estado anterior, para que, si se da una falla, se pueda volver a un estado consistente de la base de datos.

Debido a que existen dos tablas de páginas, también deben existir dos vectores de ocupación: VO para TP y VOS para TPS.

También, se tiene un *vector de situación* VS de p campos, en donde p denota el número de segmentos de la base de datos. Este vector indica a todo momento si el i-ésimo segmento se encuentra abierto $VS[i] = 1$, o si se encuentra cerrado $VS[i] = 0$.

Por ejemplo, el punto 1. de la trasferencia de páginas, se descompone asimismo en los siguientes pasos:

- Se aplica la función de apertura al segmento involucrado
- Se hace $VS[i] = 1$

- Se recopia TP en TPS
- Se envía el vector TP al buffer de tablas de páginas BTP.

A continuación, se introducen los pasos que sigue el SABD para regresar al estado anterior después de ocurrida una falla [LORI77].

Para *regresar al estado anterior* primeramente se consulta el vector de situación VS para determinar los segmentos que estaban abiertos en el momento del percance y que fueron los únicos que pudieron haber sido afectados. Para el segmento i-ésimo de la base de datos, el cual se encuentra abierto, el vector TPS refleja el último estado y debe recopiararse en TP. Luego, se debe cerrar el segmento haciendo VS[i] = 0.

Por su parte, para pasar a *un nuevo estado*, el SABD trabaja de la siguiente forma. Si se desea grabar el nuevo estado de una página que se encuentra en el segmento i-ésimo S, se utiliza el vector TP el cual tiene la información necesaria. En efecto, analizando los componentes k para los cuales $TP[k] \neq TPS[k]$, se puede actualizar el vector de ocupación VO de forma que indique que la página de datos $t = TP[k]$ está ahora ocupada y que la página de datos $s = TPS[k]$ está libre. Es posible que se pueda producir una falla cuando se está actualizando el vector de ocupación VO, lo que implicaría su destrucción. Para evitar esto, se elabora la nueva tabla en VOS y cuando esta operación se realiza, se cierra el segmento S, es decir, se hace VS[i] = 0.

Finalmente, para *rehacer el nuevo estado*, se usan indicadores de protección para liberar las páginas de datos que contienen el nuevo estado. Además, se recopia el vector VOS en VO y se cierra el segmento S, haciendo VS[i] = 0.

La técnica de paginación tiene la ventaja de que permite deshacer una transacción en una forma muy simple. Sin embargo, una de las desventajas es que el hecho de que las páginas actualizadas cambian de posición en los discos. Esto dificulta el almacenamiento de las relaciones de la base de datos en forma contigua y requiere estrategias complejas

as de páginas BTP.

os que sigue el SABD para
rida una falla [LORI77].

imeramente se consulta el
los segmentos que estaban
que fueron los únicos que
segmento i-ésimo de la base
vector TPS refleja el último
se debe cerrar el segmento

stado, el SABD trabaja de la
o estado de una página que se
liza el vector TP el cual tiene
zando los componentes k para
alizar el vector de ocupación
de datos $t = TP[k]$ está ahora
 $k]$ está libre. Es posible que se
tá actualizando el vector de
trucción. Para evitar esto, se
esta operación se realiza, se
 $[i] = 0$.

rtado, se usan indicadores de
datos que contienen el nuevo
en VO y se cierra el segmento

ja de que permite deshacer una
e. Sin embargo, una de las
ginas actualizadas cambian de
nmacenamiento de las relaciones
requiere estrategias complejas

de administración de almacenamiento. Además, cuando las relaciones son muy voluminosas, esta estrategia provoca un gran desperdicio de memoria.

11.4 TECNICA DE LA BITACORA

El mecanismo anterior introducido en [LORI77] permite proteger la base de datos en un cierto estado, pero no toma en cuenta la ejecución de transacciones concurrentes.

Si varias transacciones han actualizado una relación de un determinado segmento, el retorno a un estado anterior o el paso a uno nuevo, tendrá por efecto, o bien anular o bien confirmar en bloque todas las modificaciones de todas las transacciones. Para solventar este inconveniente se usa la técnica de la bitácora.

Las actividades de las transacciones se graban en una bitácora, -en inglés *log*- que es un archivo administrado por el SABD. Un principio básico es que se supone que una bitácora no se destruye -utilizando por ejemplo archivos espejo-.

Un registro típico de una bitácora de transacciones podría contener los campos que se muestran en la figura 11.6.

Nombre de la transacción	Direcciones del segmento + la página modificada	Valor anterior a la modificación	Valor modificado
--------------------------	---	----------------------------------	------------------

Figura 11.6. Registro tipo de una bitácora de transacciones

A continuación, se estudian las diferentes operaciones que realiza un SABD para dejar una base de datos en estado consistente [GRAY81].

En primer lugar, considerar la situación que se presenta en la figura 11.7.

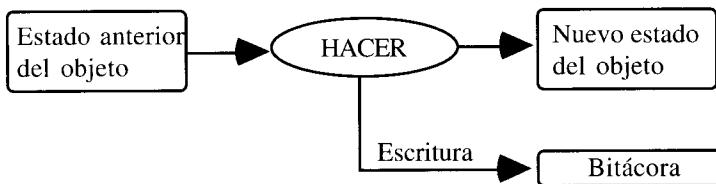


Figura 11.7 Operación HACER

En este caso, se parte de un estado anterior, se aplica la operación *hacer* -del inglés *do*- lo que provoca un nuevo estado, pero debe igualmente grabarse en la bitácora. Esta operación se considera normal y es lo que se esperaría cuando se realizan las transacciones. Sin embargo, en caso de una falla de una transacción, se aplica la operación *deshacer* -en inglés *undo*- para borrar la modificación realizada por la transacción consultando la bitácora y el nuevo estado y llegar así al estado anterior, según se aprecia en la figura 11.8.

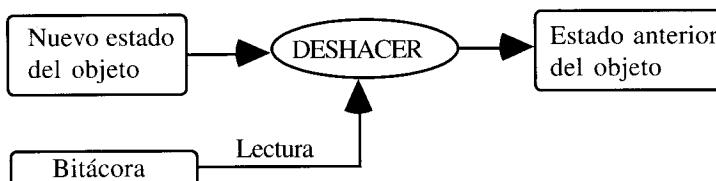


Figura 11.8 Operación DESHACER

Por su parte, en caso de falla, cuando se necesita *rehacer* -en inglés *redo*- el trabajo de una transacción, se debe partir del estado anterior y del nuevo valor almacenado en la bitácora se recrea el nuevo estado, como se aprecia en la figura 11.9.

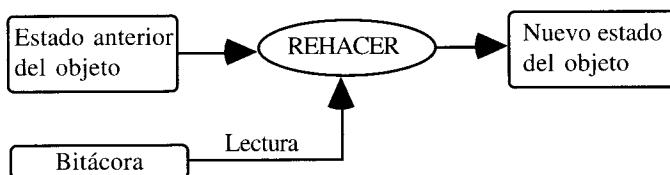


Figura 11.9 Operación REHACER

Nuevo estado
del objeto

Bitácora

CER

terior, se aplica la operación en nuevo estado, pero debe consideración se considera normal que se realizan las transacciones. Sin intervención, se aplica la operación modificación realizada por la nuevo estado y llegar así al

a 11.8.

Estado anterior
del objeto

HACER

necesita *rehacer* -en inglés *recovery*- partiendo del estado anterior y se recrea el nuevo estado,

Nuevo estado
del objeto

IACER

Puesto que la bitácora se escribe antes de proteger la base de datos, el SABD siempre puede deshacer toda actualización no confirmada que hubiese sido grabada en disco. De otro lado, si la falla se produce luego de la escritura en la bitácora en disco, la transacción podrá rehacerse usando las informaciones de la bitácora.

Cuando se realizan muchas transacciones, las modificaciones que éstas realizan aparecen en la bitácora. Entonces, cuando se presenta una falla, se requiere consultar la bitácora con el fin de analizar cuáles efectos de las transacciones deben ser eliminados o restaurados. Esta consulta a la bitácora puede requerir de mucho tiempo. Para evitar esto, se introducen *puntos de control* o de verificación -del inglés *checkpoint*- de la base de datos a intervalos regulares, los cuales pueden hacerse en cintas o bien en otros discos. Tales eventos se deben grabar asimismo en la bitácora.

Al considerar la figura 11.10 [GRAY81], se aprecian cinco transacciones que se están realizando concurrentemente. En un tiempo t_i se da un punto de control y en un tiempo t_j ocurre una falla. A partir de esta figura se van a estudiar las diferentes salidas por las que debe optar el SABD para dejar la base de datos en un estado consistente.

Cuando el SABD se pone en marcha, después de una falla, éste determina en primer lugar cuáles serán las transacciones que deben sobrevivir a la falla. Según la figura 11.10, las transacciones T_2, T_4, T_5 se confirmaron antes que ocurriera la falla. Así, el SABD debe rehacer los cambios hechos por estas transacciones.

Las modificaciones hechas por la transacción T_5 figuran en la protección, entonces T_1 no se debe rehacer.

Además, el rehacer el trabajo hecho por las transacciones T_2 y T_4 significa que debe consultarse la bitácora y recorrerse hacia adelante, desde el punto de validación.

Puesto que la bitácora contiene los nuevos valores, basta hacer cada una de las modificaciones.

Se debe observar que todas las modificaciones hechas antes de la protección figuran en la base de datos.

Por su parte, las transacciones T_1, T_5 no alcanzaron su validación y por lo tanto debe deshacerse cualquier rastro que estas transacciones hayan dejado.

El deshacer las transacciones requiere también consultar la bitácora, pero haciendo un recorrido hacia atrás hasta llegar al inicio de la transacción.

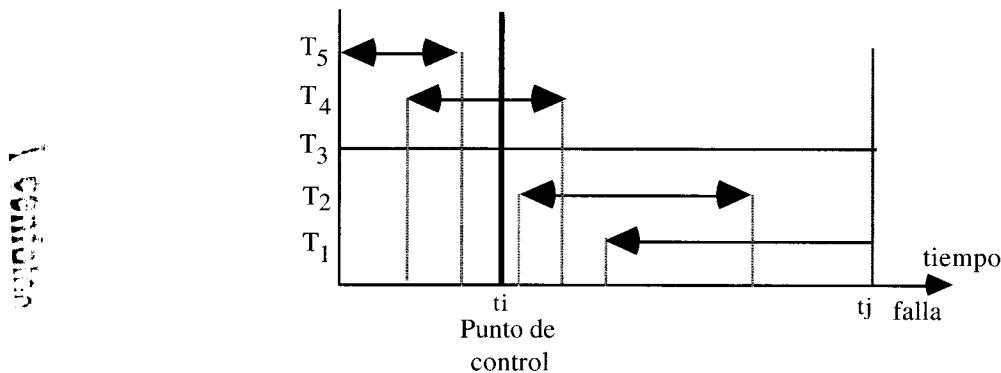


Figura 11.10 Ejemplo de transacciones concurrentes

Un SABD procede en el siguiente orden: primero deshace, luego rehace y al establecer la base de datos en un estado consistente, se graba el nuevo estado como un nuevo punto de validación.

Es importante que el trabajo hecho en el arranque pueda tolerar una falla. Con ayuda del mecanismo de protección descrito en la sección 11.2 se puede arreglar para que el estado anterior esté siempre disponible hasta que el nuevo punto de control resultante de un arranque logrado no se haya grabado. Además, con el fin de evitar los problemas ligados al acceso concurrente, el mecanismo de arranque se asegura un control exclusivo de la base de datos hasta el final del proceso de arranque.

aciones hechas antes de la
alcanzaron su validación y
tro que estas transacciones

también consultar la bitácora,
hasta llegar al inicio de la

nes concurrentes

The diagram illustrates a timeline with a horizontal arrow pointing to the right, labeled 'tiempo' (time). A vertical line segment is shown, with the label 'tj falla' (transaction j failure) positioned near its base. This visualizes the point in time at which transaction j fails.

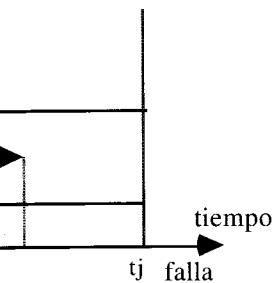
len: primero deshace, luego
n estado consistente, se graba
alidación.

el arranque pueda tolerar una
ón descrito en la sección 11.2,
esté siempre disponible hasta
de un arranque logrado no se
tar los problemas ligados al
anque se asegura un control
del proceso de arranque.

EJERCICIOS Y PREGUNTAS DE REPASO

- 11.1 Establecer y explicar los diferentes tipos de fallas que se pueden presentar en un ambiente de bases de datos.
- 11.2 Definir el concepto de página.
- 11.3 Explicar en qué consiste la técnica de paginación.
- 11.4 ¿Cómo se recupera un segmento utilizando la técnica de paginación?
- 11.5 Explicar el mecanismo de la bitácora y las diferentes operaciones que realiza un SABD bajo esta filosofía.
- 11.6 ¿Qué es un punto de control?

ficaciones hechas antes de la
no alcanzaron su validación y
astro que estas transacciones
también consultar la bitácora,
hasta llegar al inicio de la



iones concurrentes

orden: primero deshace, luego
un estado consistente, se graba
validación.

el arranque pueda tolerar una
ción descrito en la sección 11.2,
or esté siempre disponible hasta
de un arranque logrado no se
vitar los problemas ligados al
ranque se asegura un control
l del proceso de arranque.

EJERCICIOS Y PREGUNTAS DE REPASO

- 11.1 Establecer y explicar los diferentes tipos de fallas que se pueden presentar en un ambiente de bases de datos.
- 11.2 Definir el concepto de página.
- 11.3 Explicar en qué consiste la técnica de paginación.
- 11.4 ¿Cómo se recupera un segmento utilizando la técnica de paginación.
- 11.5 Explicar el mecanismo de la bitácora y las diferentes operaciones que realiza un SABD bajo esta filosofía.
- 11.6 ¿Qué es un punto de control?

12

Bases de Datos Distribuidas

*«Yo no quiero un cuchillo en manos de la patria.
Ni un cuchillo ni un rifle para nadie:
la tierra es para todos,
como el aire.*

*Me gustaría tener manos enormes,
violentas y salvajes,
para arrancar fronteras una a una
y dejar de frontera solo el aire.*

*Que nadie tenga tierra
como se tiene traje:
que tengan tierra
como tienen el aire.»*

Jorge Debravo (1938-1967),
poeta costarricense

12.1 INTRODUCCION

En los últimos 25 años, la tecnología de las bases de datos ha tenido un desarrollo vertiginoso gracias a tres eventos principales [SILB91]:

- la consolidación de los sistemas de bases de datos relacionales,
- el fortalecimiento de las técnicas para la administración de las transacciones y los mecanismos de concurrencia, y
- la investigación sobre los sistemas de bases de datos distribuidas.

El tema de los sistemas de bases de datos distribuidas, se comenzó a discutir en profundidad desde mediados de la década de 1970-1980 y los primeros prototipos aparecen a finales de esta década.

La idea detrás de un sistema de bases de datos distribuidas es desarrollar una herramienta con las características de un SABD centralizado, pero en donde los datos se pueden almacenar y accesar en diferentes sitios.

Este enfoque se justifica, en primer lugar, por un desarrollo tecnológico acelerado de las telecomunicaciones y por otra parte, por la necesidad de las empresas de descentralizar sus políticas y funciones, lo que conduce a descentralizar sus sistemas de información en diferentes ubicaciones geográficas.

A continuación se van a estudiar las características fundamentales de lo que debe ser un sistema de bases de datos distribuidas, iniciando con una introducción a las redes de computadores. También, se presenta el prototipo R*. Asimismo se discuten las ideas principales referentes a un sistema de bases de datos distribuidas heterogéneas y se presenta el sistema Multibase. Finalmente, se introduce el World Wide Web y los retos que este nuevo sistema distribuido impone a la tecnología de bases de datos.

12.2 REDES DE COMPUTADORES

Antes de entrar con mayor detalle en el concepto de base de datos distribuida, es importante introducir los conceptos más importantes del dominio de las redes de computadores.

Una *red de computadores* es simplemente un conjunto de computadores que se encuentran en diferentes sitios y que usan protocolos comunes para comunicarse entre sí. Así, los componentes básicos son los siguientes:

- un computador de tipo *servidor* que se encarga de las funciones específicas y necesarias para la red.
- varios *sitios* -que son computadores- que hacen el papel de interruptores.
- un *componente de comunicaciones*, que permite la conexión entre estos sitios.

En la figura 12.1 se presenta la arquitectura básica de una red de computadores.

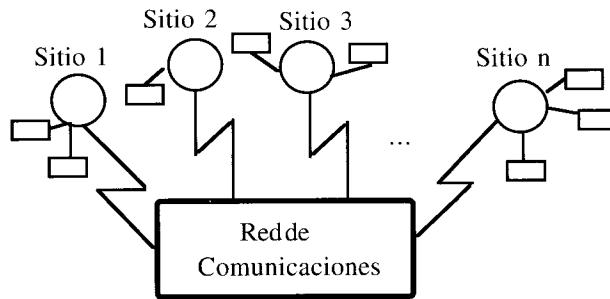


Figura 12.1 Arquitectura de una red de computadores

Las redes de computadores pueden clasificarse según varios criterios: por ejemplo, dependiendo de la distancia entre sitios, de los medios físicos de transmisión empleados, de la topología, así como de los métodos de acceso a los medios.

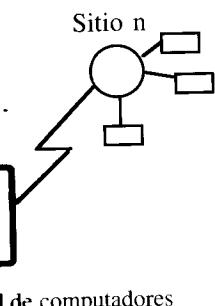
Con respecto a las distancias entre los sitios, se puede decir que una red de computadores puede ser de *área local* o *LAN* -del inglés *Local Area Network*- si las distancias entre los sitios es inferior a varios cientos de metros, quizás máximo 1 km. También, se tienen las redes de *área metropolitana* o *MAN* -en inglés *Metropolitan Area Network*- cuyas distancias son inferiores a los 50 km, por ejemplo, en la circunscripción de una ciudad. Finalmente las redes de *área*

lemente un conjunto de diferentes sitios y que usan entre sí. Así, los componentes

se encarga de las funciones

hacen el papel de interruptores. que permite la conexión entre

estructura básica de una red de



ificarse según varios criterios; a entre sitios, de los medios topología, así como de los

os sitios, se puede decir que área local o LAN -del inglés entre los sitios es inferior a 1 km. También, se tienen las en inglés Metropolitan Area ; a los 50 km, por ejemplo, en nalmente .las redes de área

amplia o WAN -en inglés Wide Area Network- para redes entre sitios con distancias superiores a los 50 km.

Con respecto a los medios físicos de transmisión empleados en las redes de computadores, se pueden citar los siguientes:

- la línea telefónica,
- el cable coaxial,
- la fibra óptica y
- el satélite.

Por su parte, con respecto a su topología, se puede hablar de:

- redes jerárquicas o de árbol,
- redes en malla,
- redes de anillo,
- redes de estrella y
- redes de bus

según se aprecia en la figura 12.2.

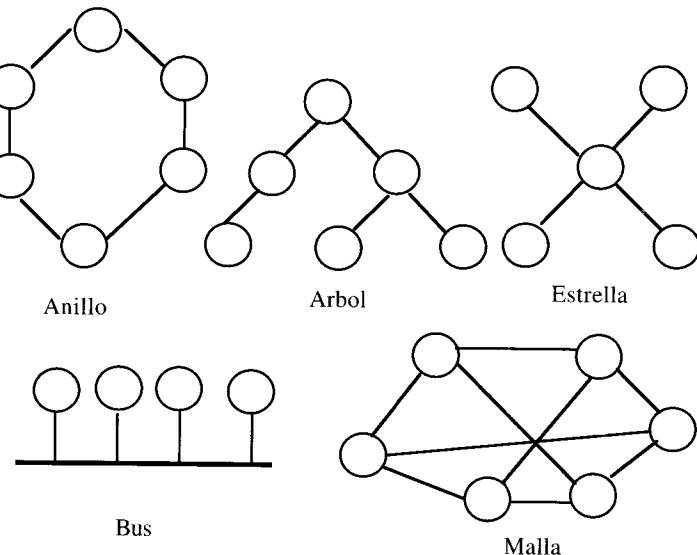


Figura 12.2 Tipos de red según la topología

12.3 COMPONENTES BASICOS DE UN SABDD

Con respecto a un sistema de bases de datos centralizado, se dijo que un SABD es un software que permite la comunicación de las transacciones de los usuarios con la base de datos; además, de los mecanismos de concurrencia, seguridad, integridad y recuperación después de fallas que permiten mantener la base de datos en un estado consistente.

Así, un SABD cuenta con dos componentes fundamentales: un *administrador de transacciones* o AT y un *administrador de datos* o AD, según se aprecia en la figura 12.3.

1 Componentes

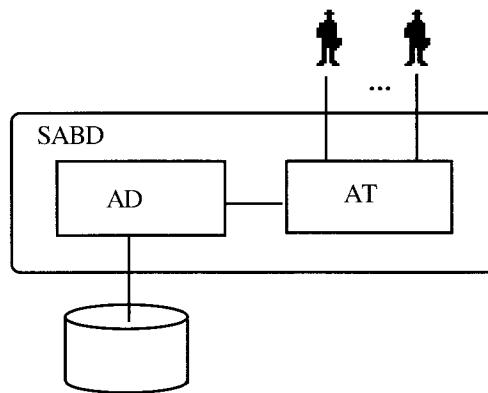


Figura 12.3. Componentes principales de un SABD

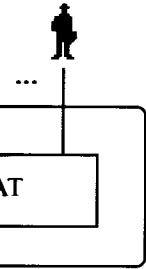
Con esta idea en mente, un *SABD distribuido* o *SABDD* no es otra cosa que un conjunto de administradores de datos y administradores de transacciones distribuidos en diferentes sitios y conectados entre ellos por medio de una red de computadores, según se aprecia en la figura 12.4, siguiendo la propuesta de [BERN81].

Las transacciones se comunican con los ATs, los cuales se comunican con los ADs que son los que administran los datos. No se pueden comunicar los ATs ni los ADs entre ellos.

N SABDD

atos centralizado, se dijo que e la comunicación de las e de datos; además, de los integridad y recuperación a base de datos en un estado

onentes fundamentales: un administrador de datos o AD,



es de un SABD

ibuido o SABDD no es otra e datos y administradores de os y conectados entre ellos según se aprecia en la figura

ATs, los cuales se comunican los datos. No se pueden

Cada transacción que se ejecuta es supervisada por un único AT, es decir, que todas las operaciones que la transacción deba hacer a la base de datos debe hacerse vía el AT asignado. Este AT controla la concurrencia.

Por su parte, los ADs administran la base de datos y en realidad vienen a ser SABDs locales.

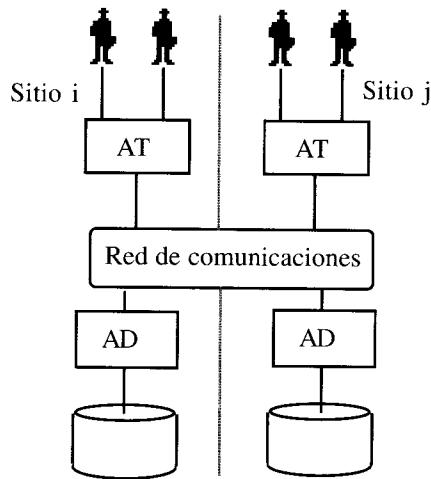


Figura 12.4 Estructura de un SABDD

Entre las funciones que realiza un AD, se encuentran las siguientes

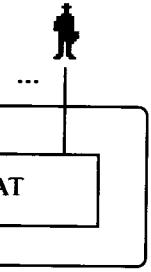
- Hacer copias de los objetos de la base de datos local en el área de trabajo del AD
- Enviar estos datos hacia el AT asignado
- Efectuar actualizaciones de los objetos
- Copiar los objetos actualizados del AT en la base de datos local

Existe una diferencia estructural entre el concepto de base de datos centralizada y base de datos distribuida. En efecto, uno de los principios de una base de datos centralizada es justamente la centralización de los datos y la no redundancia de los mismos.

JN SABDD

atos centralizado, se dijo que e la comunicación de las e de datos; además, de los integridad y recuperación a base de datos en un estado

ponentes fundamentales: un administrador de datos o *AD*,



es de un SABD

Distribuido o *SABDD* no es otra de datos y administradores de os y conectados entre ellos según se aprecia en la figura

ATs, los cuales se comunican los datos. No se pueden

Cada transacción que se ejecuta es supervisada por un único AT, es decir, que todas las operaciones que la transacción deba hacer a la base de datos debe hacerse vía el AT asignado. Este AT controla la concurrencia.

Por su parte, los ADs administran la base de datos y en realidad vienen a ser SABDs locales.

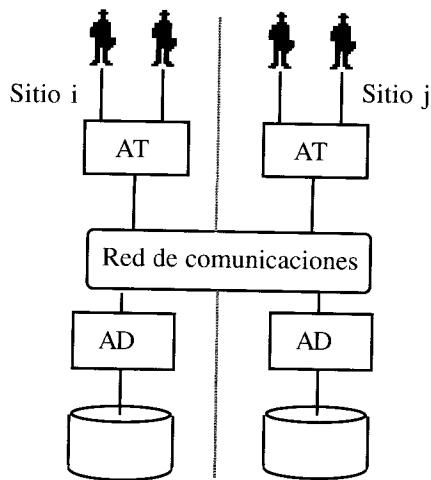


Figura 12.4 Estructura de un SABDD

Entre las funciones que realiza un AD, se encuentran las siguientes

- Hacer copias de los objetos de la base de datos local en el área de trabajo del AD
- Enviar estos datos hacia el AT asignado
- Efectuar actualizaciones de los objetos
- Copiar los objetos actualizados del AT en la base de datos local

Existe una diferencia estructural entre el concepto de base de datos centralizada y base de datos distribuida. En efecto, uno de los principios de una base de datos centralizada es justamente la centralización de los datos y la no redundancia de los mismos.

En una base de datos distribuida, por su parte, estos principios no se cumplen. En efecto, los datos se pueden encontrar distribuidos en diferentes sitios y cada uno de estos sitios pueden contar con copias del mismo objeto, o partes de él, por ejemplo, una relación. A este tipo de distribución se le llama fragmentación, concepto que será discutido en la próxima sección.

También, es importante mencionar que en los ambientes distribuidos de bases de datos, existen dos tipos de SABDDs.

Por una parte, se tiene el *SABDD homogéneo*, o simplemente SABDD, el cual se asemeja a un SABD centralizado, pero en donde los datos se almacenan y acceden desde diferentes sitios.

Por otra parte, existe el *SABDD heterogéneo* o SABDDH, que representa a un conjunto de SABDs diferentes, ubicados en varios sitios, los cuales interactúan entre sí.

12.4 MODELAJE DE BASES DE DATOS DISTRIBUIDAS

Se han desarrollado muchas metodologías para el diseño de bases de datos centralizadas. Sin embargo, las metodologías para ambientes distribuidos es casi inexistente. Una metodología de diseño interesante se puede encontrar en [ADIB78]. Por su parte, en [ANDL92], se presenta una metodología distribuida orientada a objetos.

No se puede decir que el modelaje de bases de datos centralizadas se puede extraer a las bases de datos distribuidas pues existen una serie de conceptos estructurales que hacen que los ambientes centralizados y distribuidos sean muy diferentes.

Por ejemplo, en un ambiente distribuido y por razones de eficiencia y costo, es recomendable que algunos objetos de la base de datos se encuentren duplicados en diferentes sitios. Esta idea es incompatible con el concepto de centralización única de los objetos en una base de datos centralizada, como se mencionó anteriormente.

parte, estos principios no se encuentran distribuidos en pueden contar con copias de una relación. A este tipo de concepto que será discutido en la

en los ambientes distribuidos DDs.

homogéneo, o simplemente centralizado, pero en donde los datos estén en diferentes sitios.

heterogéneo o SABDDH, que es, ubicados en varios sitios,

S DISTRIBUIDAS

s para el diseño de bases de datos. Las metodologías para ambientes heterogéneos es unaología de diseño interesante. En parte, en [ANDL92], se habla de la fragmentación de objetos.

ases de datos centralizadas se consideran pues existen una serie de ambientes centralizados y

buido y por razones de algunos objetos de la base de diferentes sitios. Esta idea es la fragmentación única de los objetos que mencionó anteriormente.

Además, las relaciones se pueden fragmentar, ya sea en forma horizontal o vertical y estos fragmentos de la relación se pueden localizar en diferentes sitios.

A continuación se profundizan estas ideas.

12.4.1 Copias de relaciones

Considerar una base de datos compuesta de m objetos, distribuidos en n sitios. Sea R una relación de dicha base de datos. Si R tiene copias en los sitios 1, ..., i , con $i \leq n$, cada una de las copias se indexa con un subíndice correspondiente al sitio. En este caso, se tendrían las copias R_1, \dots, R_i , según se aprecia en la figura 12.5

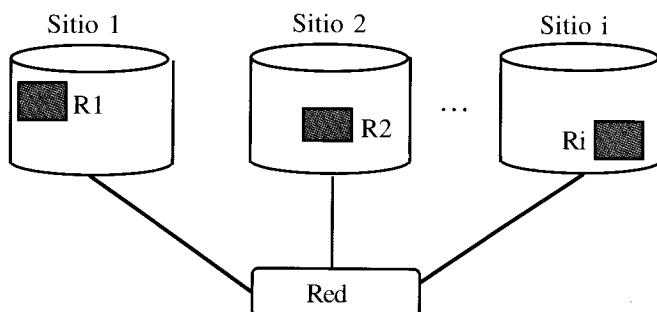


Figura 12.5 Copias de una relación en un ambiente distribuido

12.4.2 Fragmentación de relaciones

La fragmentación de una relación R puede darse en dos formas: *fragmentación horizontal* o *fragmentación vertical*.

Fragmentación horizontal

En la fragmentación horizontal, una relación R se divide en varios subconjuntos, cada uno de los cuales consiste de un determinado número de tuplas.

En realidad, una fragmentación horizontal puede verse como una selección sobre la relación R, en donde se satisface una condición.

Ejemplo. Considérese la relación SITIO. En la figura 12.6 se aprecia un posible caso de esta relación.

SITIO

Número-Sitio	Nombre-Sitio	Tipo	Continente
125	Isla Moorea	Mar/Litoral	Oceanía
126	Bahía Matsushima	Mar/Litoral	Asia
127	Irazú	Volcán	América
128	Ngorongoro	Volcán	Africa
129	Valle de la Muerte	Desierto	América
130	Kilimandjaro	Volcán	Africa
131	Bromo	Volcán	Asia
132	Atacama	Desierto	América

Figura 12.6 Ejemplo de relación

Ahora, se desea hacer una fragmentación horizontal según el tipo de sitio. En la figura 12.7 se puede apreciar dicha fragmentación, es decir, el resultado de aplicar a la relación SITIO, las siguientes tres selecciones:

```
select * from sitio where tipo = 'Mar/litoral';
```

```
select * from sitio where tipo = 'Volcán';
```

```
select * from sitio where tipo = 'Desierto';
```

SITIO-MAR/LITORAL

Número-Sitio	Nombre-Sitio	Tipo	Continente
125	Isla Moorea	Mar/Litoral	Oceanía
126	Bahía Matsushima	Mar/Litoral	Asia

Fig.12.7 continúa en la siguiente página

tal puede verse como una satisface una condición.

Figura 12.6 se aprecia un posible

	Continente
toral	Oceanía
toral	Asia
o	América
o	Africa
	América
	Africa
	Asia
	América

ción

ntal según el tipo de sitio. En la
, es decir, el resultado de aplicar
:

	Continente
itoral	Oceanía
itoral	Asia

Fig.12.7 continúa en la siguiente página

SITIO-VOLCAN			
Número-Sitio	Nombre-Sitio	Tipo	Continente
127	Irazú	Volcán	América
128	Ngorongoro	Volcán	Africa
130	Kilimandjaro	Volcán	Africa
131	Bromo	Volcán	Asia

SITIO-DESIERTO			
Número-Sitio	Nombre-Sitio	Tipo	Continente
129	Valle de la Muerte	Desierto	América
132	Atacama	Desierto	América

Figura 12.7 Ejemplo de fragmentación horizontal

Fragmentación vertical

En la fragmentación vertical, por su parte, lo que se tiene es una descomposición de una relación $R(X_1, X_2, \dots, X_n)$ en n proyecciones $R_1[X_1], R_2[X_2], \dots, R_n[X_n]$, tales que

$$R_1[X_1] * R_2[X_2] * \dots * R_n[X_n] = R(X_1, X_2, \dots, X_n)$$

Además, cada proyección $R_i[X_i]$ puede ubicarse en un sitio diferente.

Ejemplo. Siguiendo el ejemplo anterior, la relación SITIO puede fragmentarse en dos relaciones, según se aprecia en la figura 12.8 y que responden a las siguientes consultas en SQL:

```
select distinct numero_sitio, nombre_sitio from sitio;
```

```
select distinct numero_sitio, tipo, continente from sitio;
```

También se puede definir una fragmentación de tipo mixta, que es una combinación de las dos fragmentaciones anteriores.

TU
Nú
-Tu
300
301
302
303
304
SIT
Nú
-Si
125
126
127
128
129
130
131
132
133
134
135
136
VIA
Nú
-Vi
03-
04-
05-
06-
07-

SITIO-NOMBRE		SITIO-TIPO		
Número -Sitio	Nombre-Sitio	Número -Sitio	Tipo	Continente
125	Isla Moorea	125	Mar/Litoral	Oceanía
126	Bahía Matsushima	126	Mar/Litoral	Asia
127	Irazú	127	Volcán	América
128	Ngorongoro	128	Volcán	Africa
129	Valle de la Muerte	129	Desierto	América
130	Kilimandjaro	130	Volcán	Africa
131	Bromo	131	Volcán	Asia
132	Atacama	132	Desierto	América

Figura 12.8 Ejemplo de fragmentación vertical

12.5 MECANISMOS UTILIZADOS EN UN SBDD

En esta sección se discutirán los temas de procesamiento de consultas, control de la concurrencia y recuperación en un ambiente distribuido de bases de datos.

12.5.1 Procesamiento de consultas distribuidas

Un tema muy importante que involucra la tecnología distribuida de bases de datos es la referente al procesamiento de consultas distribuidas.

Para estudiar este tema considerar las tres relaciones de la base de datos del Club de Ecoturismo distribuidas en tres sitios diferentes, según se aprecia en la figura 12.9.

Par
preser

TIPO

Tipo	Continente
Mar/Litoral	Oceanía
Mar/Litoral	Asia
Volcán	América
Volcán	Africa
Desierto	América
Volcán	Africa
Volcán	Asia
Desierto	América

tación vertical

N UN SBDD

emas de procesamiento de recuperación en un ambiente

ibuidas

a la tecnología distribuida de procesamiento de consultas

tres relaciones de la base de datos en tres sitios diferentes,

TURISTA*Sitio 1*

Número -Turista	Nombre -Turista	País
300	Carlos	Costa Rica
301	Pierre	Francia
302	John	Jamaica
303	Mario	Costa Rica
304	Alí	Túnez

SITIO*Sitio 2*

Número -Sitio	Nombre-Sitio	Tipo	Continente
125	Isla Moorea	Mar/Litoral	Oceanía
126	Bahía Matsushima	Mar/litoral	Asia
127	Irazú	Volcán	América
128	Ngorongoro	Volcán	Africa
129	Valle de la Muerte	Desierto	América
130	Kilimandjaro	Volcán	Africa

VIAJE*Sitio 3*

Número -Viaje	Número -Turista	Número -Sitio	Fecha -Salida	Fecha -Llegada	Ciudad -Salida
03-96	301	125	03/03/95	03/10/96	París
04-96	303	129	07/04/95	07/14/96	Las Vegas
05-96	301	128	07/05/95	07/12/96	Dar-es-Salam
06-96	304	127	07/06/95	07/14/96	San José
07-96	302	128	07/28/95	08/13/96	Mombasa

Figura 12.9 Ejemplo de una base de datos distribuida

Para explicar la estrategia básica del procesamiento distribuido, se presenta el siguiente ejemplo

Ejemplo. Sea la siguiente consulta:

Dar la lista de los nombres de turistas costarricenses, nombres de sitios visitados, así como las fechas de salida.

```
select      nombre_turista, nombre_sitio, ciudad_salida
from        turista, sitio, viaje
where       país = "Costa Rica"
and         turista.nombre_turista = viaje.numero_turista
and         sitio.numero_sitio = viaje.numero_sitio;
```

Para responder a la consulta del ejemplo anterior, se puede seguir la siguiente estrategia.

En primer lugar, se ejecutan las selecciones locales y las proyecciones que van a estar involucradas en los join de la consulta, así como los atributos que son respuesta a ésta.

En segundo lugar, se trasladan las relaciones a uno de los sitios en donde se pueda hacer un join o aplicar una proyección en los atributos que son respuesta a la consulta.

Ejemplo. Si se aplica esta estrategia a la consulta anterior, se obtiene lo siguiente:

En el sitio 1, en donde se encuentra la relación TURISTA, se realiza la siguiente consulta

```
select      numero_turista, nombre_turista
from        turista
where       país = "Costa Rica";
```

Así, se obtiene la relación Turista(Número-Turista, Nombre-Turista). Por su parte, en el sitio 2, se realiza la consulta

```
select      numero_sitio, nombre_sitio
from        sitio;
```

con la relación SITIO(Número-Sitio, Nombre-Sitio). Finalmente, en el sitio 3, se realiza la consulta:

```
select      numero_turista, numero_sitio, fecha_salida  
from       viaje;
```

con la relación VIAJE(Número-Turista, Número-Sitio, Fecha-Salida). Las relaciones VIAJE y TURISTA se mueven al sitio 2 y se ejecuta la siguiente consulta:

```
select      nombre_turista, nombre_sitio, fecha_salida  
from       turista, sitio, viaje  
where      turista.numero_turista = viaje.numero_turista  
and        numero_sitio = viaje.numero_sitio;
```

El primer paso que se establece en esta estrategia se conoce como *reducción*. La reducción es importante en un ambiente distribuido pues es independiente de los sitios, aumenta el paralelismo y por ende la velocidad del procesamiento de consultas. Además, la cantidad de datos que deben moverse entre sitios disminuye y por consiguiente el costo de procesamiento.

Join y semi-join

El alto costo del operador join, como se vio en el capítulo 8, se hace aun más oneroso en un ambiente distribuido. En efecto, además del costo de la operación que involucra el join, es posible que se tengan que trasladar relaciones entre sitios.

Por esta razón, se define un operador que sea menos costoso en un ambiente distribuido y que tenga los mismos efectos del join.

Este nuevo operador se denomina *semi-join*. El mismo no requiere transferencia completa de una tabla y haciendo semi-joins, con un menor costo, se puede lograr el join entre relaciones que se encuentren en sitios diferentes.

En realidad, existen dos operadores semi-join, izquierdo y derecho.

El *semi-join izquierdo* se define de la siguiente forma:

$$R <* S = (R * S)[\text{Atributos de } S]$$

Un resultado equivalente al anterior es el siguiente:

$$R <_* S = S * R[\text{Atributos comunes}]$$

Por su parte, el *semi-join derecho* se define de la siguiente forma:

$$R *_> S = (R * S)[\text{Atributos de } R]$$

Según estas definiciones, lo que busca el semi-join es aumentar más los procesamientos locales, en comparación con el join y por lo tanto bajar el costo debido a que menos información debe ser transferida entre sitios.

Suponer que la relación R y S se encuentra en dos sitios diferentes S_1 y S_2 y que la relación producto $R * S$ debe ser enviada a un tercer sitio S_3 . Se puede usar el semi-join en el siguiente procedimiento:

begin

1. Luego de hacer las selecciones necesarias, calcular en el sitio S_2 , la relación $S' = S[\text{Atributos comunes}]$.
2. Enviar S' al sitio S_1 .
3. Calcular $R' = R <_* S'$ en el sitio S_1 .
4. Enviar R' al sitio S_3 .

end

Ejemplo. Siguiendo con el ejemplo anterior, considérense las consultas hechas. Luego de realizar las selecciones y proyecciones locales se obtienen las relaciones especificadas en la figura 12.10

TURISTA

Número-Turista	Nombre-Turista
300	Carlos
303	Mario

Sitio 1

SITIO

Número-
125
126
127
128
129
130

VIAJE

Número-
300
303
301
304
302

Contrar
realizando
semi-joins

Ejemplo

Dar la lista

Esta cons

VIAJE1

Número-
301
301

Para saber
siguiente -

Figura 12.10 continúa en la siguiente

siguiente:
comunes]

de la siguiente forma:

tos de R]

semi-join es aumentar más
n con el join y por lo tanto.
nación debe ser transferida

ra en dos sitios diferentes S_1
ser enviada a un tercer sitio
e procedimiento:

sarias, calcular en el sitio
nes].

onsidérense las consultas hechas.
locales se obtienen las relaciones

Sitio 1

Figura 12.10 continúa en la siguiente página

SITIO		Sitio 2
Número-Sitio	Nombre-Sitio	
125	Isla Moorea	
126	Bahía	
127	Matsushima	
128	Irazú	
129	Ngorongoro	
130	Kilimandjaro	

VIAJE			Sitio 3
Número-Turista	Número-Sitio	Fecha-Salida	
300	125	03/03/96	
303	129	04/07/96	
301	128	05/07/96	
304	127	06/07/96	
302	128	07/23/96	

Figura 12.10 Reducción de la base de datos

Contrario a la transferencia completa de las tablas a un sitio dado y realizando los joins requeridos, se considerarán los efectos de algunos semi-joins

Ejemplo. Sea la siguiente consulta:

Dar la lista de los viajes de los turistas de Costa Rica.

Esta consulta se resuelve mediante la expresión $VIAJE1 = VIAJE <_* TURISTA$

VIAJE1			Sitio 3
Número-Turista	Número-Sitio	Fecha-Salida	
301	125	03/03/96	
301	128	05/07/96	

Para saber *los sitios a los que viajaron los turistas de Costa Rica*, se puede usar la siguiente expresión: $SITIO1 = SITIO <_* VIAJE1$

SITIO1	
Número-Sitio	Nombre-Sitio
125	Isla Moorea
128	Ngorongoro

Sitio 2

Finalmente, si se desea saber *los diferentes viajes hechos*, se podría usar la siguiente expresión: VIAJE2 = VIAJE<* SITIO1

VIAJE2

Número-Turista	Número-Sitio	Fecha-Salida
301	125	{03/03/96}
303	129	{04/07/96}
301	128	{05/07/96}
304	127	{06/07/96}
302	128	{07/23/96}

Sitio 3

Esta suposición requiere la transferencia de VIAJE[Número-Turista] del sitio 1 al 3 y reduce bastante la tabla del sitio último.

Los ejemplos 1 y 2 son semi-joins que se justifican pues su beneficio es mayor que su costo. El ejemplo 3 es un semi-join que no se justifica.

12.5.2 Control de la concurrencia en BDD

A continuación se generaliza el concepto de itinerario entre varias transacciones, pero en un ambiente distribuido. Es un problema más complejo que en un ambiente centralizado, pues, desde un sitio se puede realizar una transacción y ésta puede solicitar objetos de distintos sitios e involucrar copias o fragmentos de diferentes objetos. Así, las reglas sobre el bloqueo o la comprobación de los diferentes protocolos de concurrencia deben ser redefinidos.

Transacciones e itinerarios en ambientes distribuidos

En un ambiente distribuido, una transacción puede accesar datos que se encuentran en diferentes sitios.

Sitio 2

s viajes hechos, se podría usar la
IO1

Sitio 3

IAJE[Número-Turista] del sitio 1 al
ifican pues su beneficio es mayor que
o se justifica.

DD

cepto de itinerario entre varias
tribuido. Es un problema más
lo, pues, desde un sitio se puede
licitar objetos de distintos sitios
diferentes objetos. Así, las reglas
e los diferentes protocolos de

tes distribuidos

acción puede accesar datos que

Para que una transacción distribuida llegue a su punto de validación, se deben cumplir los dos siguientes pasos:

Paso 1

Para cada copia a_i de un objeto a actualizado por una transacción, el AT respectivo envía el mensaje al AD para una actualización preliminar. El AD envía la copia a_i a la memoria interna del AD.

Paso 2

Cuando el AT es notificado por los ADs involucrados que todas las escrituras -modificaciones- preliminares se han efectuado, el AT envía la solicitud de las actualizaciones finales de las copias de los objetos actualizados por la transacción a los ADs en donde residen las copias. Cada AD responde enviando las copias a_i a su base de datos local.

La ejecución de la transacción es satisfactoria cuando todas las copias se hayan efectuado.

A continuación se define el concepto de itinerario en el contexto de un sistema de bases de datos distribuidas.

Sean k transacciones T_1, T_2, \dots, T_k , distribuidas en n sitios. Un *itinerario local en el sitio j* se define como una sucesión de acciones de las diferentes transacciones que se ejecutan en el sitio j .

Así, k transacciones en un ambiente distribuido con n sitios se pueden modelar por medio de n itinerarios locales I_1, I_2, \dots, I_n .

Sin embargo, si se aplicara algún protocolo de control de la concurrencia a estos itinerarios, no se garantizaría que la realización de las diferentes transacciones sea consistente, como lo muestra el siguiente ejemplo.

Ejemplo. Sean los dos itinerarios siguientes

I_1 en el sitio 1	I_2 en el sitio 2
$T_1 : \text{read } a_1$	$T_1 : \text{read } b_2$
$T_1 : \text{write } a_1$	$T_1 : \text{write } b_2$
$T_2 : \text{read } a_1$	$T_2 : \text{read } b_2$
$T_2 : \text{write } a_1$	$T_2 : \text{write } b_2$

En este caso los dos itinerarios locales son consistentes. Sin embargo, no existe un orden global pues en I_1 , se tiene que la transacción T_1 precede a la transacción T_2 y en I_2 , T_2 precede a T_1 .

Así, se debe dar una condición que permita que un itinerario de transacciones distribuidas sea consistente.

Se dice que la realización de k transacciones T_1, \dots, T_k , distribuidas es consistente si cumple las dos reglas siguientes:

1. Cada itinerario local I_r es consistente.
2. Existe un orden total entre las transacciones tal que:

Si $T_i < T_j$, entonces existe un itinerario consistente I_s equivalente a I_k en donde el orden entre las transacciones se mantiene en I_s .

Utilizando el concepto de conflictos entre objetos, un itinerario distribuido consistente se define de la siguiente forma.

Se dice que dos acciones a_i y a_j se encuentran en *conflicto* si ambas se efectúan sobre la misma copia (local) de un objeto y una de las acciones es de escritura -actualización-.

Un itinerario distribuido I conformado por los itinerarios locales I_1, \dots, I_n en donde se involucran las transacciones T_1, \dots, T_k se dice consistente si se puede definir un orden ' $<$ ' sobre estas transacciones tal que lo siguiente se cumple para cada par de acciones a_i y a_j en conflicto de dos transacciones diferentes T_i y T_j :

a_i precede a a_j en cada itinerario local $I_1, \dots, I_n \Leftrightarrow T_i < T_j$.

Ejemplo. Sean las tres transacciones siguientes:

T_1 : read a, write b
 T_2 : read b, write c
 T_3 : read c, write a

12.5.3

Aut
SABD
adicio

sistentes. Sin embargo, no existe una transacción T_1 que precede a la transacción T_2 y

permite que un itinerario de

acciones T_1, \dots, T_k , distribuidas es-

istentes:

acciones tal que:
ario consistente I_s equivalente
acciones se mantiene en I_s .

entre objetos, un itinerario
iente forma.

entran en *conflicto* si ambas se
n objeto y una de las acciones

por los itinerarios locales I_1 ,
acciones T_1, \dots, T_k se dice
'sobre estas transacciones tal
de acciones a_i y a_j en conflicto

$I_1, \dots, I_n \Leftrightarrow T_i < T_j$.

y los siguientes itinerarios distribuidos:

I_1		I_2	
Sitio	Ejecución local	Sitio	Ejecución local
1	$T_1: \text{read } a_1$	1	$T_1: \text{read } a_1$
	$T_2: \text{read } b_1$		$T_1: \text{write } b_1$
	$T_3: \text{write } a_1$		$T_2: \text{read } b_1$
2	$T_1: \text{write } b_1$	2	$T_3: \text{write } a_1$
	$T_2: \text{write } c_2$		$T_2: \text{write } c_2$
	$T_1: \text{write } b_2$		$T_1: \text{write } b_2$
3	$T_2: \text{write } c_3$	3	$T_2: \text{write } c_3$
	$T_3: \text{read } c_3$		$T_3: \text{read } c_3$

En este caso ninguno de los itinerarios distribuidos es consistente

En el capítulo 9 se estudió un mecanismo para el control de la concurrencia en sistemas centralizados, llamado el protocolo de dos fases. Es interesante mencionar que este protocolo es aplicable en un sistema de bases de datos distribuidas.

Así, a partir de los itineradores locales, se puede construir un itinerador distribuido.

Cada itinerador local mantiene los cerrojos sobre los objetos almacenados en los sitios respectivos, según las reglas del protocolo de dos fases y cada **read** x o **write** x se ejecuta cuando el cerrojo sobre x adecuado se ha dado y solo depende de los cerrojos que en ese momento tenga el objeto x. Así, cada itinerador local tiene toda la información que requiere para ejecutar la acción, sin necesidad de comunicarse con ningún sitio. Y lo que es más interesante es que esta filosofía se puede aplicar a un ambiente distribuido manteniéndose así la consistencia de un itinerario distribuido [BERN87].

12.5.3 Recuperación después de fallas

Aunadas a las fallas de hardware y software que se presentan en los SABDs centralizados, para el caso de los sistemas distribuidos, se deben adicionar las fallas que pueden generarse en la red.

En efecto, el funcionamiento de un SABDD depende de la capacidad que tenga la red de comunicar los diferentes sitios en donde se encuentra la base de datos distribuida.

A pesar de que las redes actuales son bastante confiables, las fallas se pueden dar. Es el caso cuando los sitios quedan particionados en dos o más grupos, de manera tal que solo es posible la comunicación entre sitios de cada grupo.

Con el fin de comprender cómo un SABDD resuelve una inconsistencia provocada por una falla, se puede introducir para ambientes distribuidos el protocolo de *validación de dos fases*, las cuales son conocidas como fase de *votación* y fase de *decisión*.

Bajo este enfoque, cada transacción global tiene un sitio que actúa como coordinador de la misma.

Según [BELL92] la idea es que el coordinador consulte si los sitios involucrados en la transacción se encuentran listos o no para validar la transacción.

Si un sitio efectúa un aborto, o no responde dentro del tiempo requerido, entonces el coordinador envía a todos los sitios la orden de proceder a un aborto. Pero si todos los sitios responden que se ha validado, entonces la transacción queda validada. En este caso se supone que cada sitio tiene su propia bitácora y por ende puede validar o desechar una transacción.

A continuación se muestran los diferentes pasos de este protocolo [BELL92]:

Protocolo de validación de dos fases

1. Cada sitio involucrado tiene derecho a manifestar un aborto o una validación
2. Una vez que se tome una decisión no se puede volver atrás
3. Si un sitio manifiesta un aborto entonces puede abortar la transacción en forma inmediata y se conoce como aborto unilateral
4. Si un sitio manifiesta una validación, debe esperar a que el coordinador emita un mensaje de validación global o aborto global

5. S
cc
6. L

Exis

En p
datos e
sitio qu

En
involuc
respald
envía la

12.5.4

El s
por IB
de estu
admini

En

En
se com
medio
denom
las tare

El s
RSS* -
asigna
de la c

Por
protoc
interb

SABDD depende de la capacidad de los sitios en donde se encuentra

stante confiables, las fallas se quedan particionados en dos o posible la comunicación entre

un SABDD resuelve una se puede introducir para actualización de dos fases, las cuales es de decisión.

bal tiene un sitio que actúa

dinador consulte si los sitios están listos o no para validar la

esponde dentro del tiempo todos los sitios la orden de sitios responden que se ha dada. En este caso se supone por ende puede validar o

tes pasos de este protocolo

dos fases
ar un aborto o una validación
e volver atrás
de abortar la transacción en
lateral
perar a que el coordinador
to global

5. Si todos los sitios emiten una validación entonces la decisión del coordinador debe ser de validación
6. La decisión global debe ser asumida por todos los sitios involucrados.

Existen dos tipos básicos de recuperación cuando se utiliza este protocolo.

En primer lugar, si el coordinador falla, el administrador de la base de datos envía órdenes de aborto o validación a los sitios que coordinaba el sitio que quedó fuera de servicio.

En segundo lugar, antes que un coordinador envíe a los sitios involucrados las órdenes de actualización, éste establece sitios de respaldo. Si el coordinador falla, uno de estos sitios toma su lugar y envía las órdenes respectivas a los sitios involucrados.

12.5.4 Sistema experimental R*

El sistema R* es un SABDD experimental que ha sido desarrollado por IBM, Almaden Research Center, en San José, California, con el fin de estudiar los problemas inherentes en un ambiente distribuido de administración de datos.

En la figura 12.11, se muestra la arquitectura del sistema R*.

En R* se tienen varias bases de datos en diferentes sitios, las cuales se comunican por medio de CICS -software de IBM que sirve como medio de comunicación entre los diferentes sitios- vía una interfaz denominada DC -Data Communication-. Además administra las E/S y las tareas y programas.

El sistema de almacenamiento utilizado en el sistema R* se denomina RSS* -Relational Storage System-, el cual administra los dispositivos, la asignación de espacio físico, los buffers de almacenamiento, el control de la concurrencia y la recuperación del sistema.

Por su parte, el administrador de transacciones -AT- suministra un protocolo de dos fases distribuido y permite la detección y solución a los interbloqueos locales, así como los interbloqueos globales.

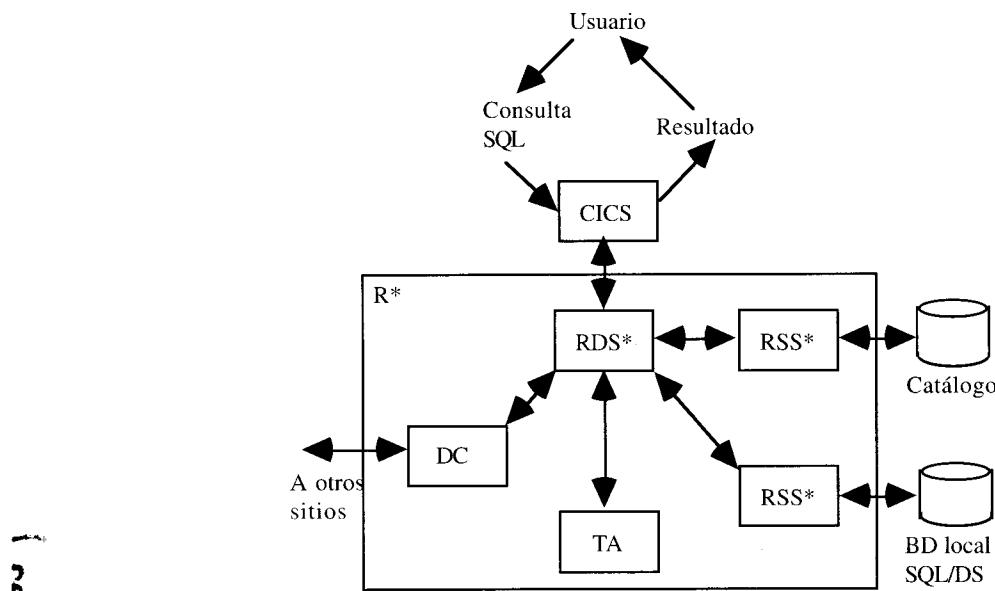


Figura 12.11 Arquitectura del sistema R*

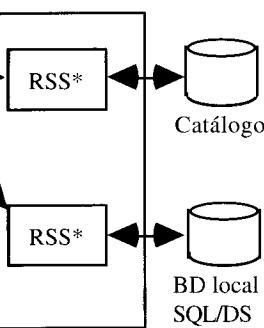
El RDS* -Relational Data System- es un procesador de lenguajes que controla la pre-compilación y ejecución de las cláusulas SQL. Este permite la creación y manipulación de las relaciones, así como la autorización y la integridad de los datos, soporta las vistas de los datos y mantiene el catálogo. Además, contiene un optimizador que escoge el acceso más eficiente a los datos.

Con respecto a la ejecución de las consultas, ésta es distribuida. El sitio de donde se hizo la consulta se convierte en el sitio coordinador.

12.6 BASES DE DATOS DISTRIBUIDAS HETEROGENEAS

En los últimos años se ha dado un gran desarrollo de la tecnología de bases de datos. Cada vez son más las organizaciones que diseñan sus sistemas de información y los implementan utilizando algún SABD. Asimismo, la gran proliferación de las redes de computadores permiten la comunicación de datos entre dos sitios cualesquiera del planeta.

do



en procesador de lenguajes que maneja las cláusulas SQL. Este maneja las relaciones, así como la vista de los datos y el optimizador que escoge el

consultas, ésta es distribuida. El se encarga en el sitio coordinador.

SISTEMAS HETEROGENEAS

desarrollo de la tecnología de organizaciones que diseñan sus sistemas utilizando algún SABD. Los sistemas de computadores permiten acceder a cualquiera del planeta.

A partir de esta explosión de información surge el deseo de establecer diálogos entre las diferentes bases de datos. Sin embargo, este es un problema difícil de resolver debido a la diversidad de:

- SABDs -INGRES, ORACLE, IMS, etc.-
- Modelos de datos -relacional, jerárquico, redes, etc.-
- Plataformas -mainframe IBM, sistema Unisys, etc.-
- Sistemas operativos -OS/VMS, Unix, etc.-

El gran objetivo de las personas que trabajan en este tipo de ambientes es desarrollar mecanismos que permitan el diálogo entre bases de datos y usuarios de forma tal que, cuando un usuario requiera consultar varias bases de datos, lo pueda lograr haciendo una sola consulta y que sea el sistema el que se encargue de buscar la información en las diferentes bases de datos para devolver una sola respuesta a la consulta planteada por el usuario. Esta situación se esquematiza en la figura 12.12.

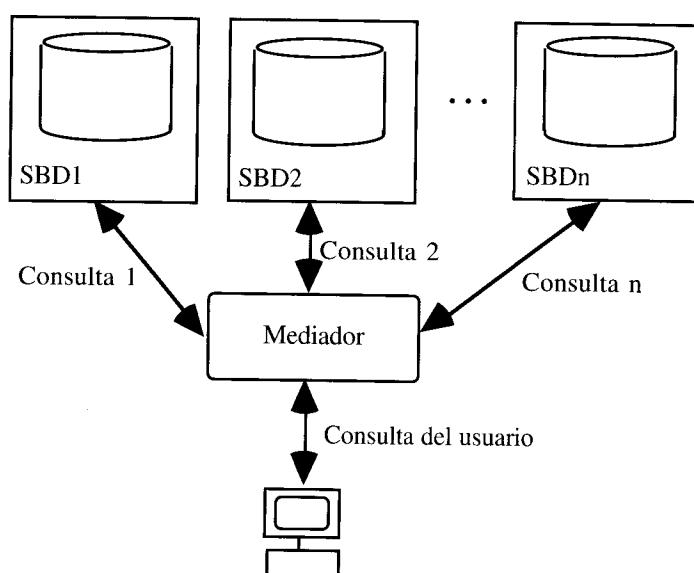


Figura 12.12 Comunicación de sistemas de bases de datos heterogéneos

Un usuario desea consultar varias bases de datos que fueron diseñadas utilizando diferentes modelos de datos e implementadas en distintas plataformas. En este caso, lo natural es que el usuario haga una sola consulta y algún módulo -que se podría denominar *mediador*- sea el encargado de interpretar dicha consulta y adaptarla a cada uno de los sistemas de bases de datos involucrados. Posteriormente, dicho módulo deberá recibir los datos deseados y presentarlos en forma que la heterogeneidad sea transparente al usuario.

Sin embargo, comunicar varios sistemas de bases de datos es un problema muy difícil, debido que se deben resolver una serie de incompatibilidades semánticas como las vistas en el capítulo 7 referente a la integración de esquemas, así como incompatibilidades sintácticas.

En esencia se puede decir que una base de datos heterogénea es un conjunto de bases de datos autónomas que pueden ser administradas en forma conjunta sin un esquema global. Por su parte, el conjunto de sistemas que permiten dicha administración se denomina *sistema administrador de bases de datos distribuidas heterogéneas -SABDDH-*.

En la próxima sección se presenta una arquitectura básica de lo que se denomina un SABDDH.

12.6.1 Arquitectura básica de un SABDDH

Muchas personas se interesan actualmente en el tema de las bases de datos heterogéneas. Varios términos se han utilizado para describir el concepto de sistemas de bases de datos heterogéneas, por ejemplo, bases de datos federadas, bases de datos múltiples, integración de bases de datos.

Varias propuestas se han hecho de SABDDHs, sin embargo, las arquitecturas son muy diversas. En la figura 12.13 se presenta una arquitectura tipo de un SABDDH.

En un ambiente de SABDDH deben instalarse copias de este software en los diferentes sitios involucrados. Si además el hardware es también diverso, se debe desarrollar para cada plataforma una versión que sea funcionalmente equivalente a la versión de las otras plataformas.

El e
distrib
de los
usuari

El e
compon
datos y
los esc
integra
model
optimiz
esto, se

El e
ANSI/S
de usu

e datos que fueron diseñadas implementadas en distintas que el usuario haga una sola denominar *mediador*- sea el adaptarla a cada uno de los posteriormente, dicho módulo entarlos en forma que la

s de bases de datos es un bien resolver una serie de as en el capítulo 7 referente compatibilidades sintácticas.

de datos heterogénea es un pueden ser administradas en su parte, el conjunto de ración se denomina *sistema us heterogéneas -SABDDH-*.

arquitectura básica de lo que

en el tema de las bases de n utilizado para describir el géneas, por ejemplo, bases de gración de bases de datos.

ABDDHs, sin embargo, las figura 12.13 se presenta una

larse copias de este software más el hardware es también forma una versión que sea las otras plataformas.

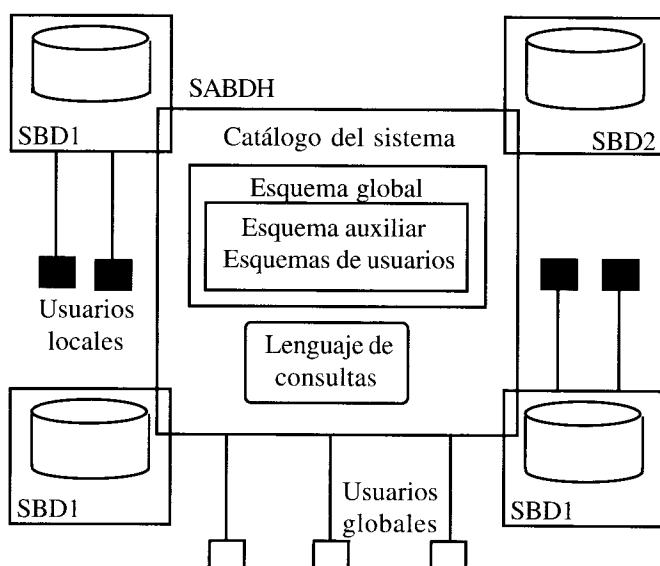


Figura 12.13 Arquitectura de un SABDDH tipo [STAN86]

El *catálogo del sistema* se puede ver como una base de datos distribuida administrada por el SABDDH, el cual incluye descripciones de los diferentes niveles de esquemas, así como descripciones de los usuarios del sistema.

El *esquema global* constituye el nivel principal de integración. Se compone de todos los objetos lógicos resultantes de la abstracción de los datos y del proceso de integración realizado sobre los datos definidos en los esquemas de bases de datos locales. Este nivel brinda una vista integrada de las bases de datos, con transparencia en la distribución y los modelos de datos. En las fases de traducción de consultas y de optimización se tiene acceso a los esquemas relacionales de base, por esto, se requiere duplicar el esquema global en todos los sitios.

El *esquema de usuario* corresponde al nivel externo de la arquitectura ANSI/SPARC. Corresponde a la definición de las vistas. Los esquemas de usuario se partitionan según el sitio del usuario. Esto impone una

restricción en el uso del sistema ya que los usuarios hacen las consultas solo desde sus sitios respectivos.

El *esquema auxiliar* se almacena en forma fragmentada en todos los sitios de la red. Se compone de los mapeos de los modelos de datos así como de información relacionada con los recursos locales y resuelve incompatibilidades de los diferentes sistemas de bases de datos. El esquema auxiliar se duplica y partitiona parcialmente en los diferentes sitios del sistema. La información duplicada caracteriza la distribución de las relaciones del esquema global y brinda la información estadística correspondiente.

Actualmente existen varios productos que se consideran SABDDH, sin embargo, ninguno es capaz de resolver todos los problemas inherentes en la integración de bases de datos, sobre todo los problemas relacionados con aspectos semánticos.

Entre los diferentes proyectos de sistemas de bases de datos heterogéneos, se pueden mencionar los siguientes:

- *ADDS* de Amoco Production Company, EE.UU. [BREI86]
- *CORDS* de IBM, EE.UU. [ATTAL95]
- *Mermaid* de System Development Corporation, EE.UU. [TEMP87]
- *Multibase* de Computer Corporation of America, EE.UU. [THOM90]
- *NDMS* de CARI, Italia [STAN86]
- *Pegasus* de Hewlett-Packard Laboratories, EE.UU. [AHME91]
- *SIRIUS-DELTA* de INRIA, Francia [STAN86]
- *UniSQL/M* de UniSQL, EE.UU. [PITO95]

En la siguiente sección se presenta el proyecto Multibase.

12.6.2 Sistema Multibase

El sistema Multibase ha sido desarrollado por Computer Corporation of America -CCA- con el fin de recuperar datos de diferentes bases de datos heterogéneas.

usuarios hacen las consultas

agmentada en todos los sitios
odelos de datos así como de
locales y resuelve incom-
de datos. El esquema auxiliar
erentes sitios del sistema. La
ucción de las relaciones del
tica correspondiente.

e se consideran SABDDH,
ver todos los problemas
s, sobre todo los problemas

emas de bases de datos
entes:

EE.UU. [BREI86]

ation, EE.UU. [TEMP87]

America, EE.UU.

es, EE.UU. [AHME91]

AN86]

95]

yecto Multibase.

por Computer Corporation
atos de diferentes bases de

En la figura 12.14 se muestra la arquitectura del sistema Multibase.

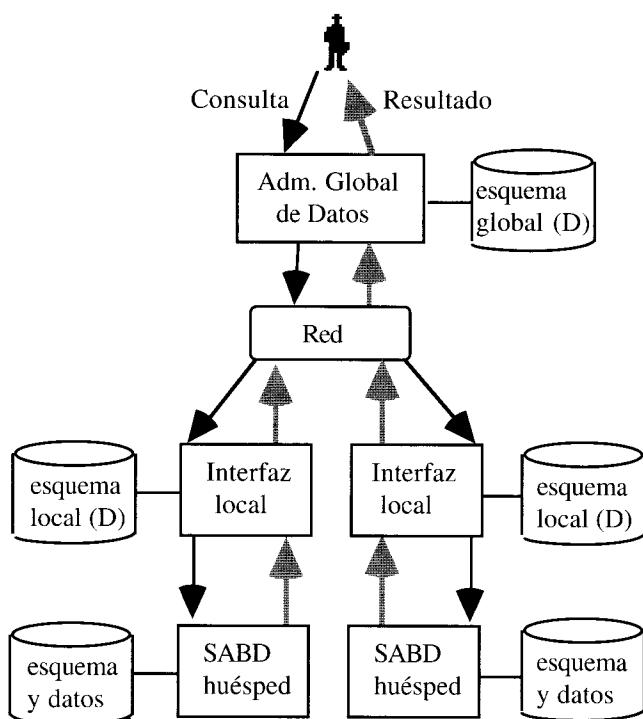


Figura 12.14 Arquitectura del sistema Multibase

Multibase soporta la integración de los modelos de datos jerárquicos, de redes y relacionales, mientras que el modelo de datos global es de tipo funcional.

Cada esquema, en cualquiera de los tres modelos de datos, debe redefinirse completamente en el modelo de datos funcional con el fin de conseguir la uniformidad entre ellos. El nuevo esquema se denomina *esquema local*.

Multibase posee el lenguaje de consultas funcional DAPLEX y se utiliza para referenciar, manipular y definir datos distribuidos, que son

vistos a través del esquema global. El esquema global se construye utilizando el modelo de datos funcional. Los esquemas locales se fusionan para constituir el esquema global que contiene la información necesaria para reconciliar inconsistencias entre los esquemas locales.

Una consulta se expresa en DAPLEX sobre el esquema global y se descompone en un conjunto de consultas locales que serán ejecutadas por los SABDs respectivos. El sitio donde se hace la consulta se convierte en el sitio coordinador. El *Administrador Global de Datos* y las *Interfaces locales* incorporan interfaces de red para establecer las comunicaciones.

El software de SABDDH es sumamente complejo. Para resolver el problema de las inconsistencias, las cuales aun no han sido resueltas completamente, se requerirán incluso de módulos de la inteligencia artificial que permita reconocer la semántica de los datos. A modo de referencia, solo el Administrador Global de Datos del sistema Multibase cuenta con 350000 líneas de programación en el lenguaje Ada [THOM90].

A continuación se muestran algunas de las interrogantes que genera la red Internet y su World Wide Web a las arquitecturas tradicionales de los sistemas de bases de datos distribuidas [SILB96].

12.7 LOS NUEVOS RETOS DEL WORLD WIDE WEB

Las arquitecturas tradicionales de los SABDDs se muestran insuficientes para apoyar las nuevas aplicaciones que se desarrollan en ambientes distribuidos. Por ello, es necesario replantearse otras estrategias que permitan responder a estos nuevos requerimientos. Un caso de éstos es el *World Wide Web*, o simplemente WWW, que está constituido por una serie de sistemas basados en hipertexto. Estos sistemas se encuentran distribuidos en todo el mundo y cada uno de estos componentes se aproxima cada vez más a un SABD.

Por la estructura misma del WWW, se tienen que redefinir una serie de situaciones de los SABDDs tradicionales. Por ejemplo, en sistemas

esquema global se construye. Los esquemas locales se que contiene la información entre los esquemas locales.

sobre el esquema global y se locales que serán ejecutadas donde se hace la consulta se Administrador Global de Datos y s de red para establecer las

e complejo. Para resolver el s aun no han sido resueltas lulos de la inteligencia artificial os. A modo de referencia, solo Multibase cuenta con 350000 HOM90].

las interrogantes que genera rquitecturas tradicionales de SILB96].

LD WIDE WEB

os SABDDs se muestran naciones que se desarrollan en cesario replantearse otras nuevos requerimientos. Un implemente WWW, que está sados en hipertexto. Estos el mundo y cada uno de estos en SABD.

nen que redefinir una serie . Por ejemplo, en sistemas

localmente autónomos, el servidor podría solicitar un pago a un cliente a cambio de un servicio brindado. Así, es imperativo implementar nuevas estrategias que permitan a un usuario pagar cierto monto por cada acceso a datos remotos.

Sin embargo, en los SBDDs clásicos los datos son un recurso privado de una organización y este problema no ha sido considerado aun.

Otro tema que debe tomarse en cuenta es el referente a la interacción entre las estrategias de procesamiento de consultas y el cobro de los servicios. Por ejemplo, si se desea consultar sobre un tema específico, una biblioteca puede brindar un servicio gratuito, pero este servicio puede que no sea tan satisfactorio como el servicio obtenido al accesar una base de datos comercial. De esta forma, el SABD debería diferenciar estos costos y hacer primero la pregunta a la fuente gratuita y luego modificar la consulta a la fuente pagada.

Otro problema que debe ser considerado con profundidad es el relativo a la seguridad de la información.

En efecto, en un sistema como el Web, es preciso desarrollar sistemas de autorización y autenticación muy flexibles, que soporten acceso según el papel jugado por el individuo. Además, se deben establecer mecanismos que soporten la venta de información a una gran cantidad de usuarios que son desconocidos al vendedor.

La naturaleza del Web presenta otros problemas. Por ejemplo, se debe tratar con [SILB96]:

- Datos cuyos esquemas no son claros, cambios espontáneos, o cuyas estructuras son irregulares
- Datos cuya definición precisa no es clara y(o) cuya integridad no es clara
- Mientras la tecnología de bases de datos ha sido muy efectiva al crear índices y otro soporte para búsquedas de información bien estructurada, es esencial que estas técnicas puedan extenderse y adaptarse al mundo no estructurado del Web.

Siempre se tiene un problema de validación del contenido de la base de datos frente a los mecanismos de entrada de datos sin integridad. Sin embargo, las nuevas aplicaciones a menudo involucran la combinación de información encontrada en diferentes recursos y estos recursos pueden ser de diferente integridad. Así, es necesario crear métodos para evaluar la integridad de tal información.

Además, las consultas deben conocer la integridad u origen de los datos. La integridad y(u) orígenes de los datos serán conceptos de gran relevancia en los futuros lenguajes de consultas.

EJERCICIOS Y PREGUNTAS DE REPASO

- 12.1 Definir los componentes principales de un SABDD
- 12.2 Explicar dos razones por las cuales es necesario contar con un SABDD
- 12.3 Explicar los conceptos de fragmentaciones horizontal, vertical y mixta.
- 12.4 Definir el concepto de transacción e itinerarios distribuidos.
- 12.5 ¿Es el semi-join un operador commutativo? Justifique su respuesta.
- 12.6 Explique la arquitectura del sistema R*
- 12.7 ¿En qué consiste una base de datos distribuida heterogénea?
- 12.8 Explicar la arquitectura del sistema Multibase.

ción del contenido de la base de datos sin integridad. Sin ello involucran la combinación de recursos y estos recursos es necesario crear métodos para

integridad u origen de los datos serán conceptos de gran utilidad.

o
e un SABDD

es necesario contar con un

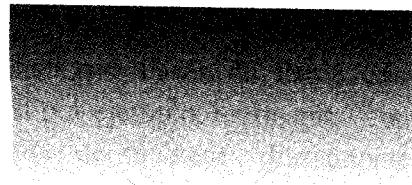
iones horizontal, vertical y

herarios distribuidos.

vo? Justifique su respuesta.

tribuida heterogénea?

ltibase.



13

Máquinas de Bases de Datos

«Vamos a comparar la actividad de nuestro intelecto en toda su plenitud, actividad que en modo alguno puede reducirse a la realización de cálculos y a la estructuración de silogismos o a la conservación de datos ya adquiridos, entonces, a mi juicio, nos formamos una impresión muy clara de que, si se excluyen algunas operaciones de carácter automático, el cerebro humano supera por muchos aspectos las más modernas máquinas y tiene posibilidades de las que adolecen las máquinas. Si esto es así, el intelecto humano es capaz de realizar una actividad que no pueden desarrollar las máquinas.»

Louis de Broglie (1892-1987),
físico francés.

13.1 INTRODUCCION

En la actualidad, es muy común encontrar bases de datos con volúmenes de varios Gb -Giga Bytes-. Así por ejemplo, un hospital con 150 camas puede generar 2 Gb de datos de imágenes diariamente [ABDE91]. El almacenamiento, recuperación y administración de bases de datos tan grandes constituyen una tarea bastante difícil. Así, se requiere de una importante transferencia de datos entre los dispositivos de almacenamiento secundario y los procesadores.

Se sabe que para una mayor eficiencia en este intercambio, se trasladan los datos empaquetados en páginas como se estudió en el capítulo 11. Sin embargo, esta práctica satura la memoria principal con datos no utilizables.

En la década de 1970, y con el fin de evitar la saturación de memorias y procesadores, surgió la idea de descentralizar el procesamiento de datos. Esto se logró trasladando una parte importante del SABD a un computador especializado, periférico del computador central, llamado *back-end*. Así, se buscaba desplazar los algoritmos de búsqueda y de actualización, lo más cerca posible de la memoria secundaria.

El back-end puede ser un computador tradicional con software específico, en este caso se habla de *computador de base de datos* o una máquina con hardware especializado para la administración de los datos en cuyo caso se habla de una *máquina de base de datos*.

El primer prototipo de máquina de bases de datos que se tiene noción se debe a los Laboratorios Bell y se bautizó como XDMS [CANA74].

Una máquina de bases de datos es un computador, también llamado *esclavo* y que puede verse como un dispositivo periférico de propósito especial para descargar el trabajo de un computador de propósito general, llamado *computador anfitrión*. Esta relación anfitrión-esclavo se puede apreciar en la figura 13.1.

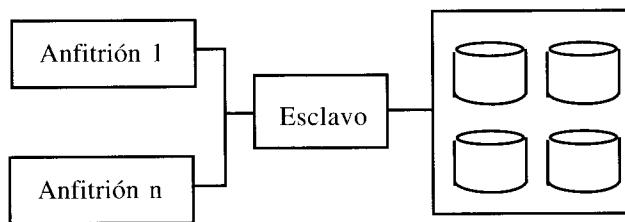


Figura 13.1 Relación anfitrión-esclavo

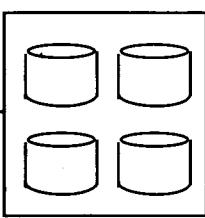
De esta manera una máquina de bases de datos está diseñada para conectarse a computadores centrales, los cuales le envían instrucciones y éstos reciben respuestas de la máquina de bases de datos. Este conjunto de instrucciones y respuestas conforman un protocolo de aplicación de la arquitectura ISO, denominado *protocolo de manipulación de datos* [GARD85b].

13.2 VENTAJAS DEL ENFOQUE MAQUINA DE BASES DE DATOS

Las máquinas de bases de datos facilitan una serie de tareas. Entre ellas, se pueden citar las siguientes:

- Solo los datos necesarios se transfieren al computador central
- El procesador central no tiene que ejecutar búsquedas ni actualizaciones
- La ejecución de las consultas es más rápida debido principalmente a la posibilidad de paralelismo, en materia de procesamiento de datos y acceso de los discos
- La seguridad mejora pues los datos se encuentran aislados de las aplicaciones
- El uso de sistemas distribuidos se favorece

Actualmente se cuenta con una gran cantidad de prototipos y productos comercializados y dos serán discutidos en la última sección del capítulo. Es difícil hablar de un estándar en materia de máquinas de



n-esclavo

de datos está diseñada para
uales le envían instrucciones
bases de datos. Este conjunto
n protocolo de aplicación de
o de manipulación de datos

UINA

an una serie de tareas. Entre

n al computador central
que ejecutar búsquedas nirápida debido principalmente
materia de procesamiento de

se encuentran aislados de las

orece

n cantidad de prototipos y
scutidos en la última sección
ar en materia de máquinas de

bases de datos. Así, algunas arquitecturas están más orientadas al software y otras por el contrario, más orientadas al hardware. Sin embargo, se tratará de establecer una arquitectura tipo, basada en la que se presenta en [GARD85b].

13.3 ARQUITECTURA DE UNA MAQUINA DE BASES DE DATOS

Partiendo de la interfaz con el usuario o los programas de aplicación, la mayoría de las máquinas de bases de datos se compone de 4 niveles fundamentales según se puede apreciar en la figura 13.2.

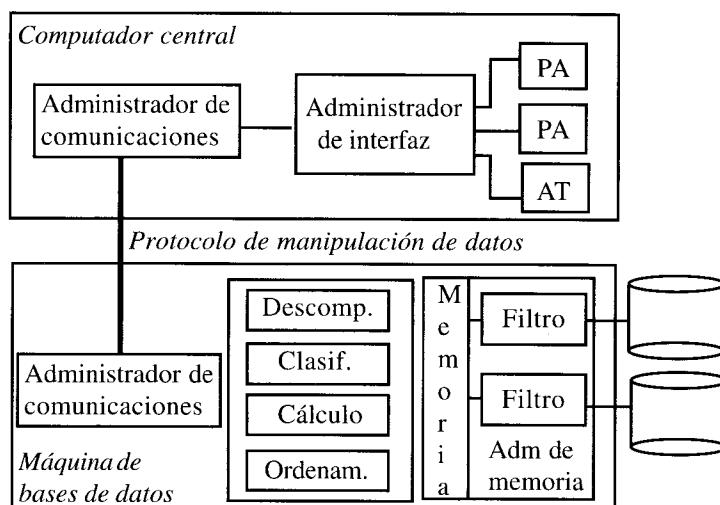


Figura 13.2 Arquitectura tipo de una máquina de bases de datos [GARD85b]

A continuación se discuten cada uno de los componentes principales que constituyen una máquina de bases de datos.

13.3.1 Administrador de interfaces

El *administrador de interfaces* es una capa externa de software que sirve para el análisis y codificación de las consultas emitidas por el

usuario. En principio, dicha capa se localiza en el computador anfitrión de la máquina de bases de datos.

13.3.2 Administrador de comunicaciones

A este nivel entra en juego el protocolo de manipulación de datos. Es decir, el administrador de comunicaciones se encarga del intercambio de datos entre el computador anfitrión y la máquina de bases de datos. Con respecto al transporte, esta capa no es específica a la administración de los datos. Pertenece en realidad al subsistema de comunicaciones.

13.3.3 Administrador-optimizador de consultas

En este componente se efectúa la descomposición y ordenamiento de instrucciones de búsqueda y de actualización en operaciones internas más simples, así como la ejecución de clasificaciones, cálculos aritméticos y administración de los caminos de acceso. También se lleva a cabo la búsqueda de éstos con un mínimo costo, para así reducir el número de entradas necesarias a la ejecución de las consultas.

13.3.4 Administrador de memoria asociativa

Una de las características importantes en un ambiente de máquinas de bases de datos es el concepto de *memoria asociativa*. Esta es una memoria que es direccionable por el contenido, es decir, es capaz de determinar si un dato está contenido en una de sus direcciones.

Pero esto es posible solo si se hace una búsqueda exhaustiva y debe hacerse en forma simultánea, es decir usar paralelismo, con el fin de obtener tiempos de respuesta aceptables.

En 1970 Slotnik [MIRA86] propone usar la rotación del disco para realizar una memoria asociativa.

El principio de una memoria asociativa se establece por medio del *direcciónamiento por contenido* que se materializa en el concepto de filtro. Un *filtro* es un procesador especializado capaz de buscar las tuplas

por m
difer
difer
perm
que v

Ex
filtro
filtro
son d

C
adopt
[MIR

La
sigui

- m
- c
- p
- u

Un
servic

- C
- C

en el computador anfitrión de manipulación de datos. Es encarga del intercambio de máquina de bases de datos. Conifica a la administración de comunicaciones.

Consultas

posición y ordenamiento de ón en operaciones internas clasificaciones, cálculos de acceso. También se lleva costo, para así reducir el de las consultas.

tiva

un ambiente de máquinas de *memoria asociativa*. Esta es una enido, es decir, es capaz de de sus direcciones.

búsqueda exhaustiva y debe paralelismo, con el fin de

ar la rotación del disco para

se establece por medio del materializa en el concepto de do capaz de buscar las tuplas

por medio de la verificación de un criterio de selección complejo en los diferentes archivos. También, administra la asignación del espacio en los diferentes discos y en general, propone métodos de acceso. Así, un filtro permite un recorrido secuencial del contenido de los datos y los datos que verifican la condición deseada constituyen el resultado.

Existen varias propuestas para los filtros. En efecto, se puede tener un filtro por cada pista del disco; un filtro por cada cabeza lectora o bien, un filtro por cada disco. La mayoría de los prototipos y productos actuales son del tipo de un filtro por cada disco.

Con respecto a la conexión de los filtros, la solución que se ha adoptado es la utilización de buffers, según se aprecia en la figura 13.3 [MIRA86].

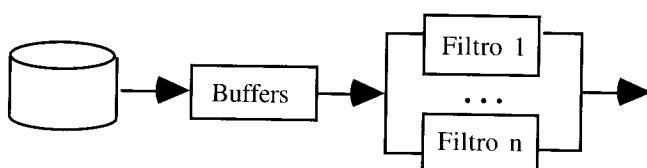


Figura 13.3 Filtrado por medio de buffers

La mayoría de las máquinas de bases de datos incluyen al menos los siguientes componentes:

- memoria caché
- controladores inteligentes
- procesadores I/O
- unidades de cintas (respaldos)

Una máquina de bases de datos realiza además, varios controles y servicios:

- Control de autorización de acceso
- Control de la integridad semántica de los datos

- Control de la atomicidad de las transacciones
- Procedimientos de recuperación después de una falla.

Una máquina de bases de datos se conecta a sistemas mainframes, minicomputadores, microcomputadores y estaciones de trabajo individuales o en red.

Resumiendo se puede decir que mientras que la máquina de bases de datos administra el almacenamiento de los datos y las funciones de recuperación, la unidad central de proceso del computador anfitrión queda libre para el procesamiento de las aplicaciones. Las máquinas de bases de datos representan una forma poderosa para administrar bases de datos voluminosas. El usuario de la máquina de bases de datos interactúa en un inicio con el computador anfitrión. Así, la solicitud de la información es enviada a la máquina de bases de datos para el procesamiento. Entonces, se examina la base de datos y la respuesta es enviada al computador anfitrión.

13.4 GRADO DE PARALELISMO DE LAS MAQUINAS DE BASES DE DATOS

Las máquinas de bases de datos son sistemas con procesadores múltiples. Mientras que la mayoría de las máquinas comercializadas ejecutan las consultas de bases de datos, una tras otra, en forma secuencial, es posible construir máquinas que ejecuten varias consultas en paralelo y que procesen varios flujos de datos en paralelo.

En arquitectura de computadores se puede hablar de cuatro tipos de computadores, según la clasificación de M. Flynn [SANS88].

- *SISD (Single Instruction Single Data)*. Estos son los computadores convencionales y conocidos como máquinas de von Neuman, o máquinas secuenciales, pues ejecutan una instrucción a la vez, con un solo procesador, según se aprecia en la figura 13.4.

ones

de una falla.

a sistemas mainframes,
estaciones de trabajo

de la máquina de bases de
datos y las funciones de
el computador anfitrión
ciones. Las máquinas de
para administrar bases de
bases de datos interactúa
Así, la solicitud de la
bases de datos para el
de datos y la respuesta es

LAS MAQUINAS DE

emas con procesadores
íquinas comercializadas
una tras otra, en forma
s que ejecuten varias
rios flujos de datos en

nhablar de cuatro tipos de
nn [SANS88].

os son los computadores
nas de von Neuman, o
instrucción a la vez, con
figura 13.4.

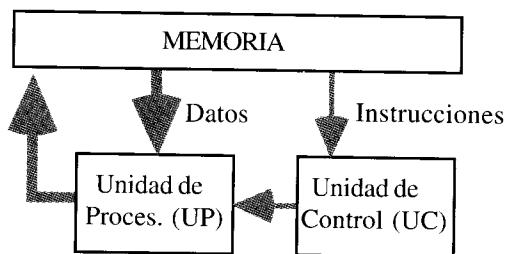


Figura 13.4 Arquitectura SISD

- **MISD (Multiple Instruction Single Data):** Estos computadores ejecutan varias instrucciones de búsqueda en forma simultánea sobre un único flujo de datos posibles. Estas máquinas son llamadas también del tipo *pipeline*. En este caso el procesador se compone de varios niveles, encargado cada uno de la parte de la instrucción proveniente de la memoria, según se aprecia en la figura 13.5. Entre este tipo de computadores se pueden mencionar el IBM 3083 y el ICL 3900.

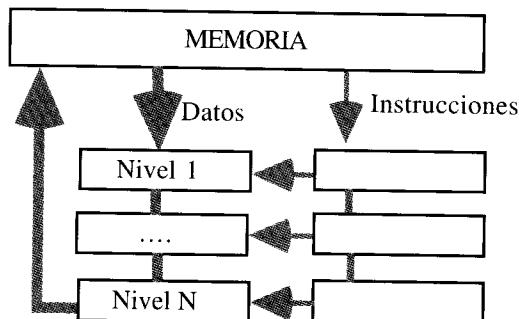


Figura 13.5 Arquitectura MISD

- **SIMD (Single Instruction Multiple Data):** Los computadores de este tipo ejecutan una misma instrucción sobre varios flujos de datos posibles. Entre estas máquinas se pueden mencionar el AMT DAP y Goodyear MPP. En la figura 13.6, se aprecia este tipo de arquitectura.

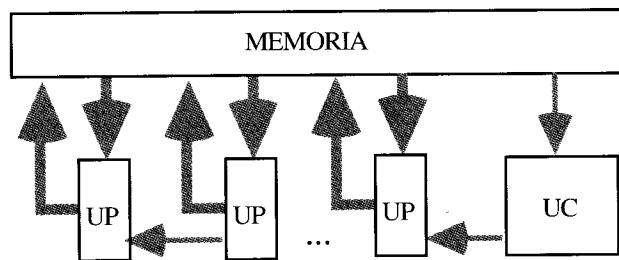


Figura 13.6 Arquitectura SIMD

- **MIMD (Multiple Instruction Multiple Data):** Estos computadores ejecutan varias instrucciones sobre varios flujos de datos. Entre este tipo de computadores se pueden mencionar a Denelcor HEP y Cray X-MP. En la figura 13.7, se muestra la arquitectura de un sistema MIMD.

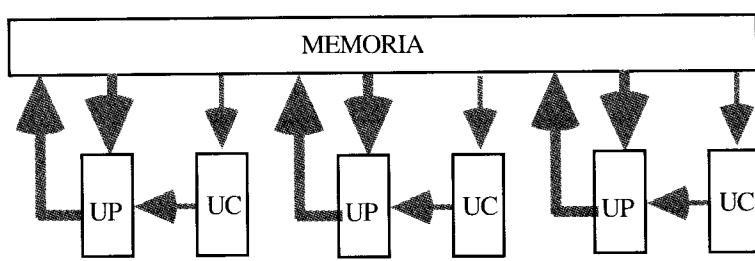


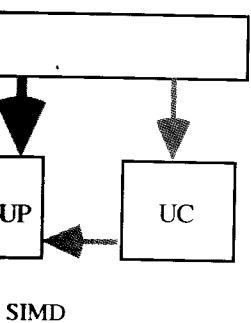
Figura 13.7 Arquitectura MIMD

13.5 PRODUCTOS COMERCIALIZADOS

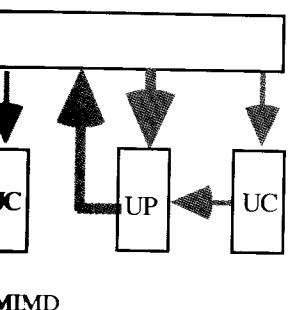
En la actualidad, muchas organizaciones que manejan grandes cantidades de datos como bancos, ministerios, centros de investigación, utilizan máquinas de bases de datos.

Entre las máquinas que se han comercializado, se encuentran las siguientes:

- **DBC/1012** de Teradata Corp.



*Data): Estos computadores
ios flujos de datos. Entre este
ionar a Denelcor HEP y Cray*
a arquitectura de un sistema



*nes que manejan grandes
os, centros de investigación,*

cializado, se encuentran las

- *SQL Mach 1* de Advanced Data Servers (ADS)
- *RDM/2* de Amperitif Corporation
- *Server/300 (700, 8000)* de Britton Lee, Inc.
- *Mega/Net 1000, 2000, 3000* de Mega/Net Co.

En menor o mayor grado, las máquinas de bases de datos son computadores con varios procesadores y hardware especializado, que administran los datos para uno o varios computadores centrales anfitriones.

Las máquinas de bases de datos disponibles hoy poseen una estructura de SABD relacional y soportan el SQL como lenguaje de consultas.

A continuación se estudiarán las arquitecturas de dos máquinas de bases de datos.

13.5.1 DBC/1012

La máquina de bases de datos DBC/1012 tiene una arquitectura MIMD y es considerada la más popular de las máquinas de bases de datos relacionales por su poder y la cantidad de unidades vendidas.

La estructura del hardware y software en esta máquina de bases de datos permite un poder en cada procesador de más de 9 MIPS -en conjunto hasta 100000 MIPS- y una capacidad de más de 2500 transacciones por segundo.

En la figura 13.8 se aprecia un esquema de la arquitectura de la DBC/1012.

Posee *unidades de disco rápidas* o DSU -del inglés Disk Storage Unit- que permiten el almacenamiento de cientos de Gb, según se requiera.

Además, integra tres tipos de procesadores [PAGE92]:

- *Procesadores de acceso* o AMP -del inglés Access Module Processor-, que forman el motor de la base de datos, y se encargan

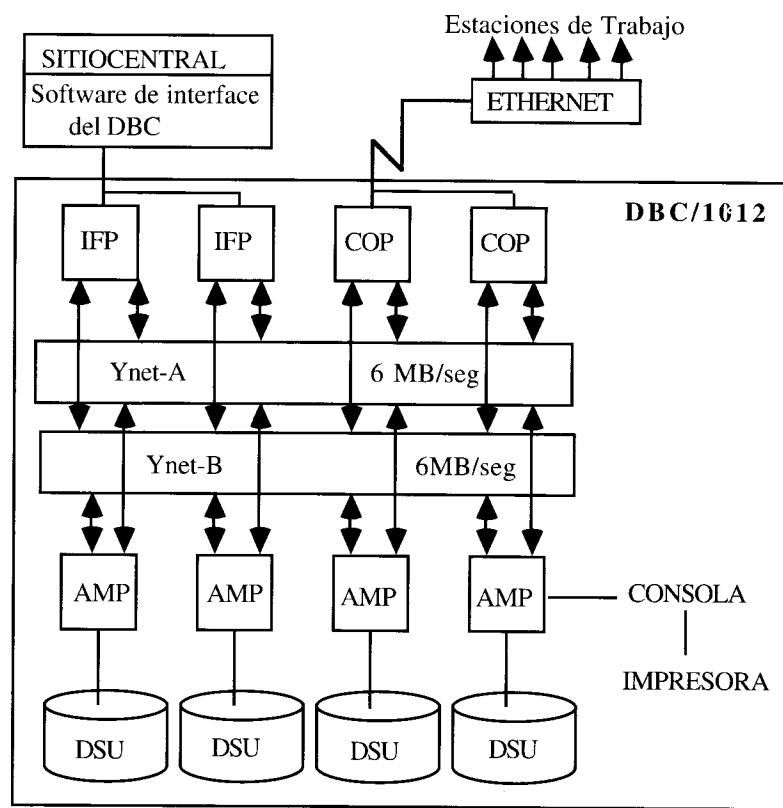
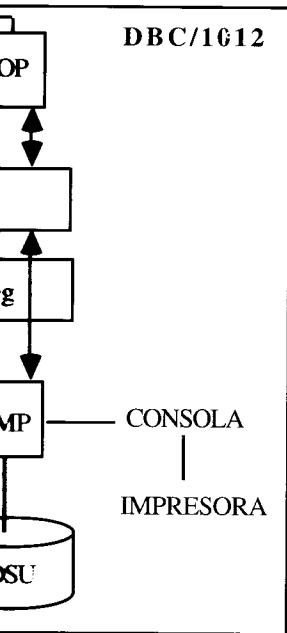
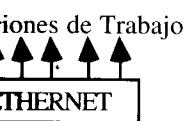


Figura 13.8 Arquitectura de la máquina de bases de datos DBC/1012

de administrar las tuplas almacenadas en los DSU asociados. Además, se encargan de todos los aspectos de acceso, búsqueda y actualización de la base de datos.

Cada AMP es multitarea y las tuplas de una relación pueden repartirse entre varios discos. Además, una consulta muy voluminosa puede monopolizar todos los AMP. Así, el grado de paralelismo está en función del trabajo que requiera la consulta.

- *Procesadores de interfaces o IFP* -del inglés Interface Processor-, que se encargan de administrar el flujo de las consultas y resultados entre la



ses de datos DBC/1012

as en los DSU asociados. Efectos de acceso, búsqueda y

as de una relación pueden además, una consulta muy los AMP. Así, el grado de que requiera la consulta.

és Interface Processor-, que se consultas y resultados entre la

DBC y los computadores centrales, entre otros los sistemas mainframes IBM que corren bajo MVS, MVS/SP, VM/CMS y TPF2; los mainframes Bull con sistemas operativos GCOS 8 y los mainframes de Unisys con OS 1100. Un IFP acepta consultas en SQL proveniente del computador central, escoge una sucesión adecuada de acciones y las envía al AMP para que se procesen, y luego envía los resultados al computador central.

- *Procesadores de comunicación o COP* -del inglés Communications Processor- que se encargan de administrar los enlaces con una red local. Un COP permite accesar la DBC/1012 por medio de microcomputadores, estaciones de trabajo UNIX usando una red LAN del tipo Ethernet.

Por su parte el *Y-Net* es un mecanismo de interconexión rápido e inteligente. Constituye el eje central de la arquitectura del procesamiento en paralelo. Está concebido para concentrar el poder de procesamiento de los procesadores en donde el número puede variar de 3 a 1024 procesadores.

13.5.2 SABRE

La máquina de base de datos llamada SABRE -Système d'Accès à des Bases de Données Relationnelles-, es un sistema que se inició como un proyecto de investigación en el INRIA -Institut de Recherche en Informatique et Automatique- de Francia.

Se puede definir como un computador con una arquitectura de tipo MIMD. SABRE se compone de un conjunto de procesadores virtuales, en donde cada tipo se encuentra asociado con un paso funcional. En la figura 13.9, se muestra la arquitectura de esta máquina.

A continuación se introducen en forma somera cada uno de los componentes de SABRE.

- *PIV (Procesador de vistas e integridad)*. Este procesador es el más externo y trabaja sobre una vista de la base de datos. Sus funciones son:
 - Controlar los derechos de acceso a la vista

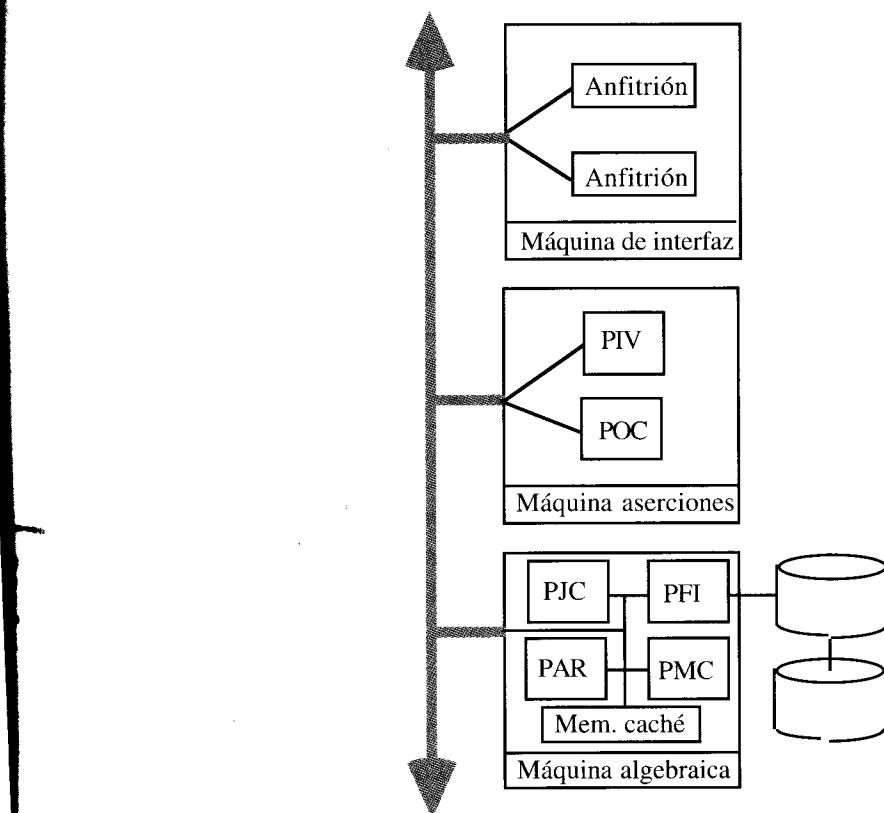


Figura 13.9 Arquitectura de la máquina de bases de datos SABRE

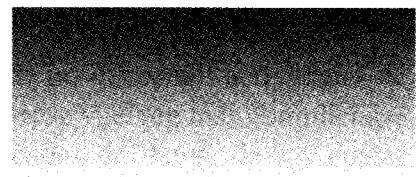
- Traducir la consulta expresada en relaciones virtuales descritas en la vista en una consulta expresada en relaciones almacenadas en la base de datos
- Controlar la integridad y actualización
- *PAR (Procesador de acceso a las relaciones)*. Trabaja sobre una relación y administra los caminos de acceso. Así, cuando se introducen tuplas, determina las particiones en donde se deben realizar dichas inserciones.

- *PJC (Procesador de joins y clasificaciones)*. Este procesador realiza joins, clasificaciones y funciones de agregación en las particiones.
- *PMC (Procesador de administración de la memoria caché)*. Se encarga de la asignación de la memoria principal y administra cuáles particiones deben almacenarse en memoria principal.
- *PFI (Procesador de filtrado)*. Garantiza la ejecución de las selecciones y actualizaciones cuando se hace la transferencia de datos entre la memoria caché y los discos.

En resumen, se puede decir que las máquinas de bases de datos surgen como una alternativa para responder en forma eficiente a las necesidades de rendimiento aceptable de bases de datos muy voluminosas debido a que los computadores tradicionales no se adaptan plenamente a la administración de grandes cantidades de datos.

EJERCICIOS Y PREGUNTAS DE REPASO

- 13.1 Definir el concepto de protocolo de manipulación de datos.
- 13.2 Determinar la diferencia entre un computador de bases de datos y una máquina de bases de datos.
- 13.3 ¿Cuáles son los componentes básicos de una arquitectura tipo de una máquina de bases de datos.
- 13.4 Explicar la clasificación de los computadores según su grado de paralelismo.
- 13.5 Explicar someramente la arquitectura de la DBC/1012.
- 13.6 Mencionar 4 máquinas de bases de datos comercializadas.



14

La Orientación a Objetos en Bases de Datos

«El espacio considerado como medio vacío, homogéneo, en el cual se dan las cosas, es una seudorrepresentación. Si suprimiéramos en nuestra representación todas las imágenes y todos los recuerdos de objetos exteriores, suprimiríamos al par el espacio. Es falsa la suposición de un espacio sin cuerpos. La noción de espacio es abstraída de los objetos de los cuerpos extensos... Tiempo y espacio, formados a posteriori, carecen de sentido cuando se les considera en abstracto; son seudorrepresentaciones; no se refieren a objeto alguno; son negaciones de objetos.»

Antonio Machado (1875-1939),
poeta y filósofo español.

14.1 INTRODUCCION

Debido a la creciente automatización de las diferentes actividades de una organización, cada día, las aplicaciones se vuelven más complicadas pues requieren manipular grandes cantidades de datos, de los cuales muchos son complejos.

Entre estas nuevas aplicaciones se puede hablar de los sistemas de diseño en ingeniería, los sistemas multimedios y el procesamiento de imágenes, así como aquellos sustentados sobre bases de conocimiento en el dominio de la Inteligencia Artificial.

Otro tipo de aplicaciones involucran la realización de transacciones de larga duración y manipulaciones millonarias de caracteres, como es el caso de bases de datos científicas.

Además, debido a que en los últimos años, la capacidad de los computadores, las redes y las herramientas de ingeniería de software, se ha mejorado constantemente, hoy se hace necesario buscar otras opciones, que permitan desarrollar sistemas cada vez más complejos y en donde los productos relacionales actuales -no el modelo relacional-, se muestran insuficientes.

En efecto, con los productos relacionales, no se pueden modelar estructuras complejas y no se permiten varios niveles de abstracción. Además, no se cuenta con mecanismos adecuados de reglas de integridad, operaciones de disparo y eventos.

Una de estas alternativas parece ser la llamada orientación a objetos. Si bien es cierto este enfoque no es nuevo, pues se puede encontrar en lenguajes de programación como Simula y Smalltalk, es en los últimos años que se ha visto la aplicabilidad de este paradigma en la tecnología de bases de datos.

Sin embargo, esta aplicabilidad no ha estado exenta de problemas y confusión, pues aun no existe en el medio, consenso en lo que debe ser un SABD orientado a objetos y cuáles sus características mínimas.

Al estudiar en las próximas secciones algunas arquitecturas de productos y prototipos orientados a objetos, se podrá evidenciar esta falta de estandarización.

14.2 PARADIGMA DE ORIENTACION A OBJETOS (OO)

En un ambiente OO, el software se organiza como un conjunto de objetos discretos que incorporan tanto su estructura como su comportamiento, en contraste con ambientes tradicionales en donde estas dos características se encuentran separadas.

Los primeros intentos en aplicar la orientación a objetos al ambiente de bases de datos fue mediante la construcción de interfaces como una capa externa a los SABDs relacionales.

Uno de los problemas que se tienen actualmente es la falta de estándares en la concepción y definición de términos. Sin embargo, a continuación se van a introducir los conceptos mínimos, que se consideran fundamentales en la aplicación de este paradigma a la tecnología de bases de datos.

14.2.1 Objetos

Los datos en la orientación a objetos se cuantifican en entidades discretas y distinguibles llamadas objetos. Un *objeto* representa una cosa, concreta o abstracta, que es un elemento de una aplicación. Cada objeto se caracteriza por una descripción y un comportamiento. La descripción se hace a partir de los valores que este objeto toma de un conjunto de *atributos* y su *comportamiento* a partir de un conjunto de *operaciones*.

Por su parte, los estados de un objeto, es decir, los diferentes valores que toma un objeto desde su creación hasta su destrucción, se manipulan por medio de *métodos*, los cuales son invocados por las operaciones.

Ejemplo. Los siguientes conceptos pueden representarse como objetos de un sistema.

- una
- un a
- una
- el p
- un e

Un c
de dat
concurr

Es i

Cad:
transpa

Al i
tener c
objetos

Un
tambié
O2 [DE

14.2.2

Com
objeto

Por
la mis
como

El
arbitra
represe
describ

s algunas arquitecturas de s, se podrá evidenciar esta

OBJETOS (OO)

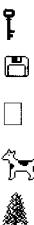
aniza como un conjunto de su estructura como su tes tradicionales en donde adas.

cación a objetos al ambiente ión de interfaces como una

actualmente es la falta de e términos. Sin embargo, a nceptos mínimos, que se ón de este paradigma a la

se cuantifican en entidades Un *objeto* representa una tónto de una aplicación. Cada y un comportamiento. La que este objeto toma de un a partir de un conjunto de

dicir, los diferentes valores u destrucción, se manipulan dos por las operaciones. presentarse como objetos de un

- una llave
 - un archivo de personal
 - una relación de una base de datos
 - el perro de Carlos
 - un árbol conífero
- 

Un objeto puede ser *concreto* -es el caso de una relación en una base de datos- o *conceptual* -es el caso de un protocolo de control de la concurrencia-.

Es importante establecer que en el paradigma OO *todo es objeto*.

Cada objeto posee un identificador interno que se llamará *oid* y es transparente al usuario.

Al identificarse cada objeto internamente en forma única, se pueden tener dos objetos con características idénticas y considerarlos como objetos diferentes. Este no es el caso de los sistemas relacionales.

Un *oid* puede identificar al objeto durante toda su existencia o también solo en el transcurso de una sesión, como es el caso del sistema O2 [DEUX91], el cual será estudiado en la próxima sección.

14.2.2 Clases de objetos

Como se vio anteriormente, cada objeto posee un *oid*, es decir, un objeto tiene una existencia que es independiente de sus valores.

Por otra parte, los objetos se pueden reunir en conjuntos, que tengan la misma estructura y comportamiento. Dichos conjuntos se conocen como *clases de objetos*.

El definir un conjunto de objetos como una clase es un hecho arbitrario y depende en gran medida de la aplicación que se esté representando. Una clase de objetos, a diferencia de una relación, puede describir un conjunto infinito de objetos.

Ejemplo. En la figura 14.1, se presentan varios objetos que son árboles. Estos, a su vez podrían ser considerados dentro de una clase de objetos que podría llamarse simplemente *Arbol*.

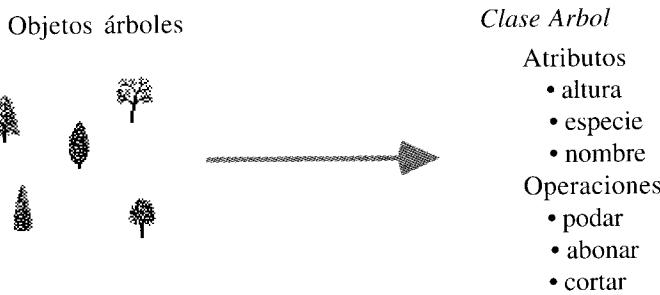


Figura 14.1 Ejemplo de clase de objetos

Otro hecho importante es que un atributo de un objeto, a su vez, puede ser un objeto, lo que permite un manejo de objetos de complejidad arbitraria.

Además, una misma operación puede ser compartida por varias clases de objetos. Esto se conoce como *polimorfismo*. En efecto, en el mundo real, una operación es una abstracción de un comportamiento análogo de diferentes tipos de objetos.

A modo de comentario, según la mayoría de los autores, en el caso del modelo relacional, se afirma que los objetos -llamados tuplas- tienen componentes atómicos y no se permite que un atributo, a su vez, sea una relación. Sin embargo, se han hecho estudios con relaciones que no cumplen esta característica y que se denominan *NF2* -del inglés *No First Normal Form*- [MAKI77].

Así, el establecer una biyección entre el concepto de clase de objetos y relación es incorrecto. En esta línea, y en consonancia con el concepto verdadero de relación, en [DARW95] se considera que una clase de objetos debe identificarse en el mundo relacional, no con una relación pero sí con el concepto de dominio.

De
-nada
objeto:
estable
domin

Así
estable
model

14.2.3

Otr
de abs
esenc
lo tan
relativ

As
un sis
sin t
imple

Mu
ejemp
de un

Si
son m
mejor

La
son l
el cap

A
se di

etos que son árboles. Estos, a su vez, son objetos que podrían llamarse

Clase Arbol

Atributos

- altura
- especie
- nombre

Operaciones

- podar
- abonar
- cortar

objetos

de un objeto, a su vez, puede ser un objeto de complejidad

compartida por varias clases. En efecto, en el mundo

los autores, en el caso del *relational model*. Los -llamados tuplas- tienen un atributo, a su vez, sea una tupla o no, con relaciones que no cumplen la *normalización NF2* -del inglés *No First Normal Form*.

El concepto de clase de objetos tiene una resonancia con el concepto de clase de tipos. La diferencia es que una clase de tipos es conceptual, no con una relación

De esta forma, si se pudieran establecer dominios personalizados -nada en el modelo relacional impide esta afirmación-, una clase de objetos como AVE, podría definirse con atributos como *vuelo del ave* establecido sobre un dominio *video*, el atributo *canto del ave* sobre un dominio *sonido*, etc.

Así, se estaría sacando el modelo relacional de los límites establecidos por los productos desarrollados y no por limitaciones del modelo mismo.

14.2.3 Abstracción

Otro concepto fundamental en el paradigma orientado a objetos es el de abstracción. En efecto, la *abstracción* hace surgir las características esenciales de un objeto -que lo distinguen de otros tipos de objetos- y por lo tanto procura fronteras conceptuales rigurosamente definidas, relativas al punto de vista del usuario.

Así por ejemplo, el uso de la abstracción en una fase de análisis de un sistema significa tratar con conceptos del dominio de la aplicación, sin tomar en cuenta las decisiones posteriores de diseño e implementación.

Muchos de los sistemas que se desean representar son complejos. Por ejemplo, el sistema circulatorio del cuerpo humano, el establecimiento de una sociedad, etc.

Sin embargo, las posibilidades humanas para manejar dichos sistemas son muy limitadas. De ahí, el uso de la abstracción, que permitirá una mejor comprensión de un sistema.

Las dos herramientas fundamentales para el manejo de la abstracción son la herencia y la composición, conceptos que fueron introducidos en el capítulo referente al modelo de datos semántico.

Además, cuando una clase de objetos es subtipo de varios supertipos, se dice que se tiene una *herencia múltiple*.

14.2.4 Encapsulamiento -Ocultamiento- de la información

El principio de *encapsulamiento* traduce el hecho de que el acceso a un módulo sea solamente por medio de su interfaz externa. Esto separa la implementación de un módulo, que es visible solo a quien lo implementó, de su interfaz, que describe la forma en que los usuarios pueden ver dicho módulo.

En el paradigma OO, la unidad de modularidad es el objeto y por lo tanto puede ser sujeto de encapsulamiento. Es el procedimiento por el cual se ocultan todos los detalles de un objeto que no forman parte de sus características esenciales. Así, se separan los aspectos externos de un objeto, que son accesibles a otros objetos, de los detalles de implementación del objeto, que se ocultan a los otros objetos.

La abstracción y el encapsulamiento son conceptos complementarios [BOOC91]. En efecto, la abstracción se concentra en la vista externa de un objeto mientras que el encapsulamiento niega a las personas ver en el interior de un objeto,

Ejemplo. Considerar el objeto PACIENTE que aparece en la figura 14.2.

En este caso, varios atributos a su vez son objetos, como por ejemplo, Historial médico del paciente, así como información médica del padre y de la madre del paciente. En este caso se desea recibir información sobre las enfermedades infecciosas del paciente, así como las que padecieron sus padres. El cómo se ha implementado esta información, queda completamente oculto al usuario.

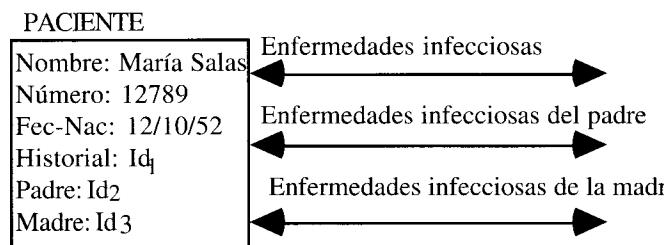


Figura 14.2 Ejemplo de encapsulamiento

e la información

El hecho de que el acceso a la interfaz externa. Esto separa visible solo a quien lo forma en que los usuarios

laridad es el objeto y por tanto. Es el procedimiento un objeto que no forman se separan los aspectos s a otros objetos, de los que se ocultan a los otros

conceptos complementarios entra en la vista externa de llega a las personas ver en el

rece en la figura 14.2. s. como por ejemplo, Historial ca del padre y de la madre del ación sobre las enfermedades eron sus padres. El cómo se ha ente oculto al usuario.

eciosas
eciosas del padre
eciosas de la madr
amiento

14.2.5 La persistencia

La *persistencia* es la propiedad de un objeto a través de la cual su existencia trasciende el tiempo -el objeto continúa existiendo luego de que su creador haya dejado de existir- y(o) el espacio -el lugar en donde el objeto fue creado cambia de dirección-.

Resumiendo, se puede decir que el paradigma OO se basa en 5 conceptos fundamentales [BERT91]:

- Cada entidad del mundo real se modela como un objeto y a cada objeto se le asocia un único identificador que se denota por *oid*.
- Cada objeto posee atributos -estructura- y operaciones -comportamiento-. El valor de un atributo puede ser un objeto o un conjunto de objetos lo que permite la definición de objetos complejos.
- Los valores de los atributos representan el estatus del objeto, que se accesa o modifica enviando mensajes al objeto para invocar las operaciones correspondientes.
- Los objetos comparten los mismos atributos y operaciones y se agrupan en clases de objetos.
- Una clase de objetos puede definirse como una especialización de una o más clases de objetos -llamadas sub-clases- y hereda los atributos y operaciones de sus super-clases.

14.3 SABDs ORIENTADOS A OBJETOS Y RELACIONALES AMPLIADOS

A pesar de que se habla mucho de productos de cuarta generación -sistemas orientados a objetos-, aun existe cierta inmadurez en este campo.

En primer lugar, se nota falta de estandarización en los productos. Es por esta razón que, en Estados Unidos, Francia, etc., se han constituido

grupos de universitarios e investigadores para definir un estándar. Por ejemplo, a nivel de modelos de datos OO, se encuentra una gran gama de productos, cada uno de ellos con una interpretación muy particular de los conceptos tales como objetos, clases de objetos y de enlaces lógicos -herencia, composición, asociación, etc.-.

En segundo lugar, se puede afirmar que, un software que rompe o provoca cambios revolucionarios en un campo dado, tarda algunos años antes de presentarse como una alternativa real. Hay que recordar que los SABDs relacionales tardaron 20 años en imponerse.

Los SABDs relacionales, como sistemas maduros constituyen la mejor de las escogencias que se puedan hacer actualmente para cualquier aplicación de tipo administrativo. Sin embargo, con lo que respecta a otros tipos de aplicaciones, los sistemas relacionales son inadecuados. Por esta razón, grupos de investigadores y desarrolladores de software se han dado a la tarea de establecer nuevos SABDs capaces de subsanar estas deficiencias.

Así, en estos momentos se pueden distinguir dos grandes tendencias: ampliar los SABDs relacionales o bien desarrollar nuevos SABD-OO.

La primera tendencia se refiere a la idea de construir un SABD con base relacional, pero permitiendo extensiones con el fin de responder a los requerimientos de aplicaciones no administrativas. Es el caso del sistema *Postgres* que es una ampliación del sistema relacional *Ingres*.

En segundo lugar, se parte del principio de que, independientemente de las extensiones que se hagan de los SABDs actuales o futuros, llegará siempre a un punto en donde las funcionalidades serán insuficientes para cubrir tales requerimientos. Por eso, se propone encontrar la otra corriente de investigación que considera la pertinencia de romper con el pasado y abocarse a la elaboración de sistemas OO que se podrían llamar sistemas *puros*. Estos sistemas se construyen sobre un lenguaje de programación OO, adicionándole las características pertinentes de la tecnología de bases de datos.

para definir un estándar. Por se encuentra una gran gama interpretación muy particular de objetos y de enlaces lógicos

e, un software que rompe o tpo dado, tarda algunos años al. Hay que recordar que los ponerse.

as maduros constituyen, la er actualmente para cualquier argo, con lo que respecta a elacionales son inadecuados. desarrolladores de software se SABDs capaces de subsanar

uir dos grandes tendencias: arrollar nuevos SABD-OOs.

e construir un SABD con una s con el fin de responder a los ativas. Es el caso del sistema relacional *Ingres*.

de que, independientemente ABDs actuales o futuros, se las funcionalidades sean ientos. Por eso, se puede que considera la pertinenciaaboración de sistemas OO y sistemas se construyen sobre onándole las características os.

Sin embargo, es importante manifestar que, debido a la falta de una estandarización de los conceptos de la OO aun no se ha definido un modelo de datos tipo, que permita una evolución sostenida y homogénea de estos sistemas.

Se prevé que en los próximos años se llegue a un consenso y a la definición de estándares en los SABD-OOs que permitirá a los diferentes usuarios de una base de datos contar con una herramienta que reúna la potencia y rapidez de los sistemas jerárquicos y la simplicidad de los sistemas relacionales.

14.4 SABD-OO

La primera vez que se habla del concepto de un SABD-OO se remonta al año 1983 con una propuesta de G. Copeland y D. Maier, en la cual sugieren construir tal SABD a partir del lenguaje de programación Smalltalk [COPE84] y con la colaboración de miembros del Oregon Graduate Center. De este prototipo se desarrolla un producto y se comercializa en 1988 por Servio Logic, bajo el nombre de *GemStone* [BANC90b].

Por otra parte, en ese mismo año, la compañía Hewlett Packard implementa un proyecto bajo el nombre de *Iris*.

A pesar de que aun no existe un consenso de lo que debe ser un SABD-OO, existen algunas características mínimas que deben ser satisfechas. En efecto, un SABD para ser considerado con una etiqueta de orientación a objetos, debe satisfacer al menos los siguientes criterios:

- debe ser un SABD en el sentido tradicional
- debe ser un sistema orientado a objetos.

De esta forma, un SABD-OO debe satisfacer las reglas que aparecen en la figura 14.3 [BANC90b]:

Reglas de bases de datos

El sistema debe:

- Asegurar la persistencia de los datos.
- Poder administrar en forma eficiente una jerarquía de memorias.
- Permitir a los usuarios manipular los datos en forma concurrente.
- Permitir al usuario consultar la base en forma natural y sencilla.
- Permitir la administración de objetos complejos.

Reglas de orientación a objetos

- Los objetos deben tener una identidad independiente de su valor.

El sistema debe además:

- Permitir la noción de encapsulamiento.
- Agrupar los objetos en clases o poder establecer tipos.
- Definir una jerarquía de clases o de tipos.
- Permitir la redefinición de las operaciones.
- Permitir la programación completa de las aplicaciones.
- Permitir la extensión de las clases o tipos predefinidos por parte del usuario.

Figura 14.3 Características de un SABD-OO

A continuación, se introducen algunos prototipos y productos comercializados con una etiqueta de orientación de objetos.

Para comprender mejor el alcance de estos SABDs, se introduce un esquema que será utilizado en los ejemplos de las secciones subsiguientes.

Ejemplo. En el esquema de la figura 14.4 se aprecia una herencia formada de tres clases de objetos, en donde SITIO es la super-clase y SITIO-NATURAL y VESTIGIO-HISTORICO son las sub-clases. Además, las clases SITIO y SITIO-NATURAL contienen, respectivamente, las clases AEROPUERTO-SALIDA y CIUDAD-CERCANA.

14.4.1

El si
GIP. Al
primer
program
se haría

En la
y los di

En p
objetos,

- Un
ad
inc
- Un
rec

una jerarquía de memorias.
datos en forma

n forma natural y
complejos.

independiente de su valor.

establecer tipos.
pos.
iones.
las aplicaciones.
pos predefinidos por

SABD-OO

os prototipos y productos
ación de objetos.

tos SABDs, se introduce un
ejemplos de las secciones

recia una herencia formada de tres
per-clase y SITIO-NATURAL y
demás, las clases SITIO y SITIO-
ases AEROPUERTO-SALIDA y

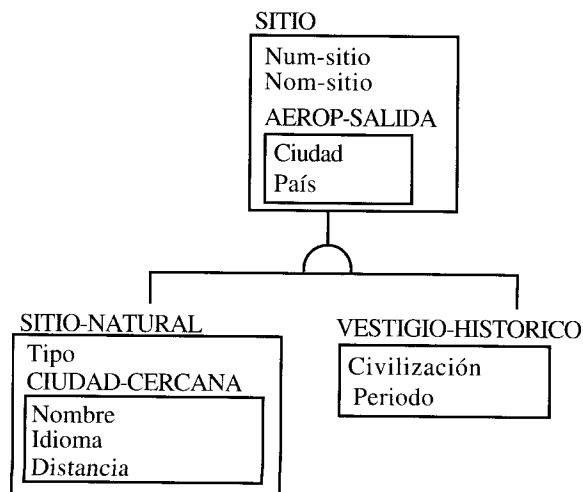


Figura 14.4 Ejemplo de una herencia

14.4.1 El sistema O2

El sistema *O2* es un SABD-OO desarrollado en Francia, por el grupo GIP Altaïr y dirigido por F. Bancilhon. El proyecto inició en 1986 y el primer prototipo se implementó utilizando los lenguajes de programación C y Lisp. Posteriormente, el producto que se comercializó se haría en C++ y Motif.

En la figura 14.5 se presenta la arquitectura funcional del sistema *O2* y los diferentes módulos que lo conforman.

En primer lugar se tiene un motor de bases de datos basado en objetos, denominado *motor_O2*. Este a su vez, se compone de:

- Un *Administrador de discos*, que se encarga de las entradas/salidas, administración de los buffers, así como del agrupamiento y la indización.
- Un *Administrador de esquemas*, encargado de la creación, recuperación, actualización y supresión de las clases, operaciones

y nombres globales. También, es responsable de la verificación de la consistencia de los esquemas.

- Un *Administrador de objetos*, encargado de administrar los objetos complejos y el envío de mensajes.

Así, el motor_O2 es la suma de un sistema de bases de datos y un sistema orientado a objetos.

Herramientas	C++	C
Generador de interfaces	Lenguaje de consultas	4GL CO₂
Administrador de esquemas		Administrador de objetos
Administrador de disco		Motor O₂

Figura 14.5 Arquitectura funcional del sistema O2

Por otra parte, se tiene un conjunto de *herramientas* que permiten un ambiente de programación gráfica. Estas herramientas permiten al programador editar y consultar los datos y los esquemas. También, brinda herramientas que permiten simplificar el trabajo de programación.

El *CO₂* es un lenguaje del tipo 4GL pues permite al usuario realizar tres tipos de tareas: programación, manipulación de bases de datos y generación de interfaces con el usuario.

El *lenguaje de consultas* es del tipo SQL, ampliado con el fin de permitir el manejo de valores y objetos complejos. Es un subconjunto del CO₂, pero puede utilizarse en forma independiente para realizar consultas del tipo *ad-hoc* o servir para llamada de los lenguajes C y C++.

nsable de la verificación de
lo de administrar los objetos

ema de bases de datos y un

C
4GL
CO ₂
nistrador
jetos
Motor O₂

del sistema O₂

rramientas que permiten un
herramientas permiten al
y los esquemas. También,
mplificar el trabajo de

s permite al usuario realizar
lación de bases de datos y

QL, ampliado con el fin de
nplejos. Es un subconjunto
independiente para realizar
amada de los lenguajes C y

Finalmente, se tiene el *generador de interfaces de usuario* llamado *Looks*, que permite el despliegue y manipulación de objetos complejos y de multimedios.

Ejemplo. Si se desea definir, utilizando el lenguaje CO2, las diferentes clases de objetos de la herencia sobre los sitios, se puede hacer de la siguiente forma:

```
add class Sitio
  type tuple (num_sitio: integer,
              nom_sitio: string,
              aeropuerto_salida: set (tuple (ciudad: string,
                                              pais: string)))
```

```
add class S_natural
  type tuple (tipo: string,
              c_cercana: set (tuple (nombre: string,
                                     idioma: string,
                                     distancia: integer)))
```

```
add class V_historico
  type tuple (civilizacion: string,
              periodo: string)
```

```
add class S_natural inherits Sitio
add class V_historico inherits Sitio
```

En este caso, la palabra **inherits** traduce el hecho de que SITIO-NATURAL y VESTIGIO-HISTORICO son subclases de la clase SITIO.

Por su parte, si se desea responder a la consulta:

Dar la lista de los nombres de los sitios y ciudades canadienses de las que pueden ser visitados.

Utilizando el lenguaje CO2, se puede responder de la siguiente forma:

```
select tuple (sitio: s.nom_sitio, aeropuerto_salida: a.ciudad)
from s in Sitio,
     a in s.aeropuerto_salida
where a.pais = 'Canadá'
```

14.4.2 El sistema GemStone

El sistema *GemStone* fue desarrollado por Servio Logic Development Corporation. El sistema originalmente fue implementado usando el lenguaje Pascal y luego fue reescrito en el lenguaje C. Se comercializa desde 1988. El modelo de datos es el mismo que el del lenguaje de programación Smalltalk. En la figura 14.6 se presenta la arquitectura cliente/servidor del sistema GemStone.

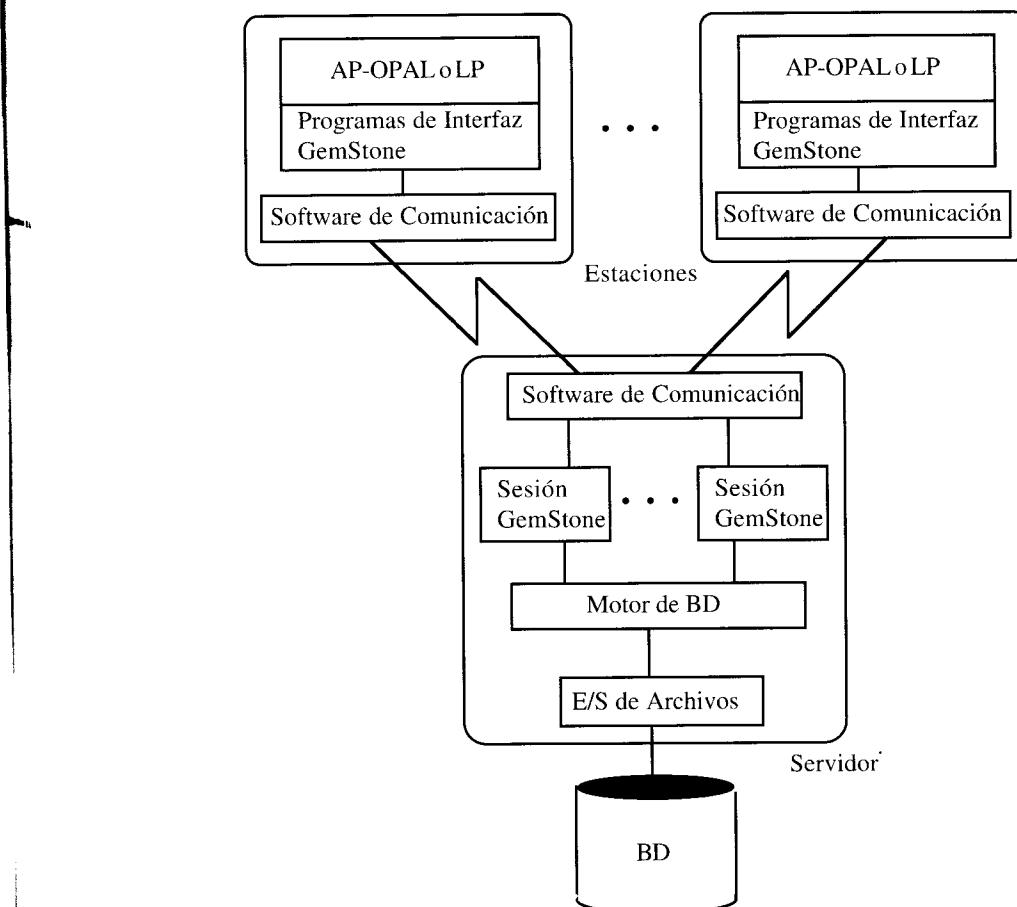
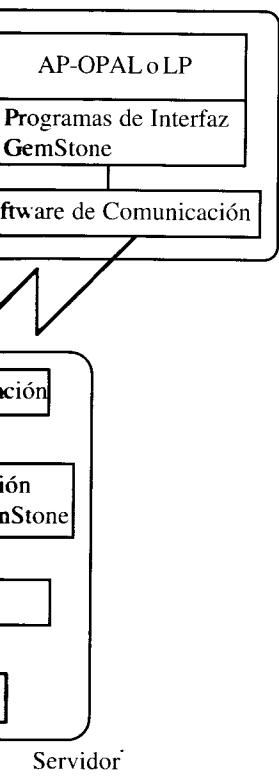


Figura 14.6 Arquitectura del sistema GemStone

Servicio Logic Development
implementado usando el
lenguaje C. Se comercializa
mismo que el del lenguaje de
se presenta la arquitectura



GemStone

Los componentes de base de la arquitectura GemStone son: los procesos *Gem* y *Stone*.

El *proceso Stone* es, en esencia, un motor de bases de datos y se encarga de la asignación de los identificadores de objetos, denominados *OOP* -del inglés *object-oriented pointer*-. Asimismo, se encarga de la asignación de las páginas en disco y administra el control de la concurrencia, la recuperación después de fallas, así como las autorizaciones. El motor reside en el servidor y accesa la base de datos por medio de llamados al sistema operativo.

GemStone tiene un lenguaje para la definición y manipulación de datos, denominado *OPAL*.

OPAL es un lenguaje de consultas y de programación muy cercano a la filosofía del lenguaje Smalltalk-80.

Por otra parte, los *procesos Gem* -conocidos también como sesiones GemStone-, brindan la compilación de los programas OPAL, las autorizaciones de los usuarios, así como un conjunto predefinido de clases y operaciones de OPAL para ser utilizados por el programa del usuario. Para cada aplicación de cliente existe un proceso Gem.

Por su parte, los *programas de interfaz GemStone* residen en las estaciones de trabajo y se encargan de efectuar las operaciones de E/S y otras funciones de interfaz.

El sistema permite además, de las aplicaciones escritas en OPAL -AP-OPAL en la figura 14.6-, el uso de lenguajes de programación como Smalltalk, C, C++, Pascal y ADA. Además, se puede interactuar con bases de datos de tipo SQL.

En el siguiente ejemplo, se muestra la definición y manipulación de datos utilizando OPAL.

Ejemplo. Sean las mismas clases de objetos del ejemplo anterior. La definición de tal entidad, utilizando el lenguaje OPAL, se da a continuación:

```

Object subclass 'Sitio'
instVarNames: #(‘num_sitio’, ‘nom_sitio’, ‘aeropuerto_salida’)
classVars: #()
immutable: false
constraints: # [ # [ num_sitio, SmallInteger ],
                  # [ nom_sitio, String ],
                  # [ aeropuerto_salida, Aeropuerto_salida ] ]

Object subclass 'Aeropuerto_salida'
instVarNames: #(‘ciudad’, ‘pais’)
classVars: #()
immutable: false
constraints: # [ # [ ciudad, String ],
                  # [ pais, String ] ]

Sitio subclass ‘S_natural’
instVarNames: #(‘tipo’, ‘c_cercana’)
classVars: #()
immutable: false
constraints: # [ # [ tipo String ],
                  # [ c_cercana C_cercana ] ]

Object subclass: ‘C_cercana’
instVarNames: #(‘nombre’, ‘idioma’, ‘distancia’)
classVars: #()
immutable: false
constraints: # [ # [ nombre String ],
                  # [ idioma String ],
                  # [ distancia String ] ]

Sitio subclass: ‘V_historico’
instVarNames: #(‘civilización’, ‘periodo’)
classVars: #()
immutable: false
constraints: # [ # [ civilización String ],
                  # [ periodo String ],
                  # [ distancia SmallInteger ] ]

```

Por su parte, si se desea responder a la consulta:

Dar la lista de las ciudades cercanas a los sitios naturales en donde se habla español y las cuales se encuentran a menos de 50 kilómetros.

sitio', 'aeropuerto_salida')

allInteger],
ing],
ida, Aeropuerto_salida]]

J.

ercana]]

distanzia')

J.
d.
g]]

o)

ing],
].
Integer]]

sitos naturales en donde se habla
50 kilómetros.

Se haría de la siguiente forma:

```
C_cercana select : { :unCiu | (unCiu.idioma = 'español') &  
                      (unCiu.distancia <50) }
```

La creación de clases en GemStone solo permite representar herencias simples, debido a que el mensaje **subclass** solo puede ser enviado a una sola clase.

14.4.3 El sistema ObjectStore

El sistema *ObjectStore* provee un lenguaje de interfaz con características similares a las de un SABD tradicional, tales como almacenamiento persistente, manejo de transacciones, acceso distribuido a los datos y consultas relacionales. Se accesa mediante el lenguaje C++, pero también se puede hacer mediante el lenguaje C.

Con respecto a la arquitectura de ObjectStore, se puede decir que se basa en una del tipo cliente/servidor. El cliente solicita páginas al servidor en respuesta a las generadas por la aplicación.

El servidor sólo trata con páginas y el cliente con objetos. Debido a que las colecciones son objetos, éstas son manipuladas por el cliente, así todas las consultas se ejecutan en el cliente [OREN92].

Ejemplo. Siguiendo con la herencia sobre los sitios, la definición de tales clases, se hace utilizando los constructores *class*, que es una generalización del *struct* del lenguaje de programación C. Así las clases del ejemplo se definen de la siguiente forma:

```
class Sitio  
{  
public:  
    int num_sitio;  
    string nom_sitio;  
    os_Set<Sitio*> aeropuerto_salida;  
}
```

```
class Aeropuerto_salida
{
    string: ciudad;
    string: pais;
}
class S_natural: public Sitio
{
    string tipo;
    os_Set<Sitio*> c_cercana;
}

class C_cercana
{
    string: nombre;
    string: idioma;
    int: distancia;
}
class V-historico: public Sitio
{
public:
    string: civilización;
    string: periodo;
}
```

Por su parte, una consulta en ObjectStore es una expresión especificada por un operador de consulta. Una expresión de consulta evalúa una colección, un objeto o un operador booleano.

Por ejemplo, la siguiente consulta:

Dar los nombres de los sitios ubicados en Nicaragua.

Se haría de la siguiente forma:

```
all_sitio
[: aeropuerto_salida -> sitio &&
aeropuerto_salida [: pais = 'Nicaragua' :] :];
```

En este caso [::] es el operador de consulta.

14.4.4 El sistema ORION

El sistema *ORION* es un proyecto que inició a finales de 1985 en el *Advanced Computer Technology Program* en MCC, Austin, Texas y ha sido comercializado bajo el nombre de ITASCA.

En la figura 14.7 se presenta la arquitectura de los módulos fundamentales que constituyen el sistema ORION.

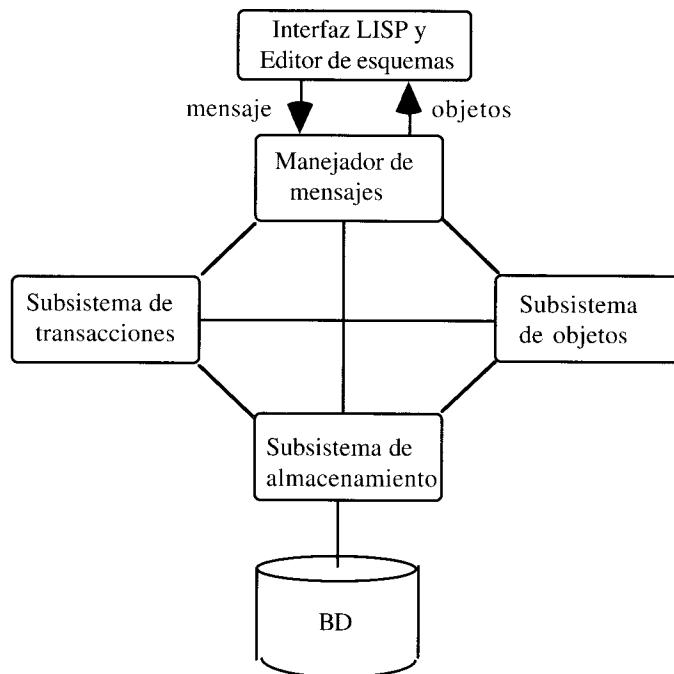


Figura 14.7 Arquitectura del sistema Orion

El *manejador de mensajes* es el encargado de recibir los mensajes que son enviados al sistema.

El *subsistema de objetos* se encarga de implementar las consultas, la evolución del esquema, el control de las versiones, administrar la bitácora y soportar objetos complejos.

Por su parte, el *subsistema de transacciones* brinda los mecanismos necesarios para mantener la integridad de la base de datos, permitiendo la realización concurrente de transacciones.

Con respecto al *subsistema de almacenamiento*, éste brinda el acceso a los objetos que conforman la base de datos. Además, administra la asignación de las páginas en los discos y permite el intercambio de los objetos entre la memoria secundaria y los buffers.

Ejemplo. Si se considera la herencia de clase de objetos que se definió en el ejemplo anterior, la definición de tal entidad, utilizando ORION, se da a continuación:

```
make-class Sitio
  :atributes [num_sitio: int,
             nom_sitio: string,
             aeropuerto_salida: Aeropuerto_salida]

make-class Aeropuerto_salida
  :atributes [ciudad: string,
             pais: string]

make-class S_natural
  :superclasses Sitio
  :atributes [tipo: string,
             c_cercana: C_cercana]

make-class C_cercana
  :atributes [nombre: string,
             idioma: string,
             distancia: int]

make-class V-historico
  :superclasses Sitio
  :atributes [civilización: string,
             periodo: string]
```

Por su parte, si se desea responder a la siguiente consulta:

es brinda los mecanismos base de datos, permitiendo

ento, éste brinda el acceso os. Además, administra la mite el intercambio de los fers.

objetos que se definió en el utilizando ORION, se da a

da]

ulta:

Dar los nombres de los sitios que se ubican en Panamá.

Se haría de la siguiente forma:

Sitio select :S nom_sitio

(:S aeropuerto_salida **some** :U (:U pais = 'Panamá'))

14.4.5 El sistema ONTOS

El sistema *ONTOS*, que es el sucesor de VBase, es un producto concebido por C. Harris de la compañía Ontologic, un proyecto que se inicia en 1989.

Se implementó en C++ y se presenta como un sistema de bases de datos distribuido del tipo cliente/servidor.

ONTOS brinda un almacenamiento de objetos persistente para programas en C++. Está integrado con el lenguaje por medio de la introducción de las clases definidas como subclases de *Entity*.

Por su parte, los objetos se dan en dos estados: ya sean *desactivados*, almacenados en los discos y *activados* como objetos ordinarios de C++ en un programa de aplicación.

Con respecto al modelo de datos, se inspira de C++ y brinda dos interfaces de programación.

Por una parte, se tiene un lenguaje de consultas del tipo SQL. Este consiste en ampliar el tipo de argumentos permitidos en la cláusula *select from where* del SQL.

Por otra parte, se tiene el propio lenguaje C++, el cual se utiliza para escribir los métodos de las clases de objetos.

También, ONTOS permite varias versiones de un objeto dado.

A continuación se estudian algunos productos que han seguido la estrategia de ampliar un sistema relacional para permitir el uso de objetos complejos.

14.5 SABDs RELACIONALES AMPLIADOS

14.5.1 El sistema Postgres

El proyecto *Postgres* -Post Ingres- inicia en 1986 como una extensión del SABD Ingres. Ha sido escrito en el lenguaje de programación C y consta de 180000 líneas de código [STON91].

Postgres se apoya en la idea de extender el modelo relacional con mecanismos generales que permitan el soporte de objetos complejos, así como implementar jerarquías de objetos. Entre los mecanismos, se pueden mencionar los *tipos de datos abstractos*, los *datos de tipo procedimiento* y las *reglas*.

Una base de datos en Postgres se puede ver como un conjunto de tablas. El tipo de un atributo puede ser *atómico*: entero, punto flotante, booleano, date, o bien *estructurado*: arreglos o procedimientos.

Entre los objetivos principales del sistema Postgres se pueden mencionar los siguientes:

- mejorar la administración de los objetos complejos
- permitir la definición de nuevos tipos de datos, nuevos operadores y nuevos métodos de acceso
- brindar mecanismos primarios para la definición de bases de datos
- mejorar la seguridad de funcionamiento

Ejemplo. Si se desea definir el esquema de los sitios, utilizando el lenguaje Postquel, se haría de la siguiente forma:

```
create Sitio
  (num_sitio = int4,
   nom_sitio = char[30],
   ubicacion = Ubicacion)

create Aeropuerto_salida
  (ciudad = char[20],
   pais = char[20])
```

S

1986 como una extensión
lenguaje de programación C y

el modelo relacional con
de objetos complejos, así
entre los mecanismos, se
factos, los *datos de tipo*

ver como un conjunto de
co: entero, punto flotante,
o procedimientos.

Sistema Postgres se pueden

complejos
datos, nuevos operadores

definición de bases de datos

sistios, utilizando el lenguaje

```
create S_natural
  (tipo = char[20],
   c_cercana = C_cercana)
inherits (Sitio)
```

```
create c_cercana
  (nombre = char[30],
   idioma = char[20],
   distancia = int4))
```

```
create V_historico
  (civilización = char[30],
   periodo = char[30])
inherits (Sitio)
```

Por su parte, si se desea responder a la siguiente consulta:

Dar los nombres de todos los sitios con un número menor que 160.

Se haría de la siguiente forma:

```
retrieve (S.nombre)
from E in Sitio*
where S.num_sitio < 160
```

En este caso el * después de Sitio indica que la consulta recorrerá la clase Sitio, así como todas las subclases, en este caso S_NATURAL y V_HISTORICO.

14.5.2 El sistema Starburst

El proyecto *Starburst* se inició en 1985 en el centro IBM Almaden Research Center, en San José, California.

Inicialmente se desarrolló como una ampliación del sistema relacional R, pero como este no fue pensado para que evolucionara, el primer intento fue un fracaso.

Starburst inicialmente fue escrito en lenguaje de programación C para equipos IBM PC y soporta todos los estándares del lenguaje SQL.

Así, se puede decir que Starburst es un sistema relacional al que se le han agregado componentes del paradigma de orientación a objetos y de la inferencia lógica.

Del paradigma orientado a objetos, se tiene:

- Objetos
- Encapsulación
- Identificadores de objetos
- Jerarquías de tipos
- Herencia.

Por su parte, de la lógica hereda lo siguiente:

- Recursión en SQL
- Funciones
- Reglas definidas por el usuario
- Operaciones de disparo.

El sistema Starburst consiste de un procesador de consultas denominado *Corona* y de un administrador de datos llamado *Core*.

En la figura 14.8 se muestra la arquitectura de Corona [HAAS90].

En el procesamiento de consultas en Corona se tienen 5 fases: el parsing y la verificación semántica, la optimización de la reescritura de la consulta, la optimización de planos, el refinamiento de la consulta y la evaluación de la consulta.

Además, en Corona se utilizan dos tipos de estructuras de datos principales: el modelo de grafo de la consulta -QGM- que es una red semántica interna que describe la consulta en la compilación y el plano de evaluación de la consulta o plano optimizado, que es la

salida ejecuta

Por garantiz



Plan

TIEM
TIEM

14.5.3

Final
híbrido
relacio

Este
experi

ma relacional al que se le
orientación a objetos y de

salida de la fase de compilación y se usa en tiempo de corrida para ejecutar la consulta.

Por su parte el sistema Core almacena y recupera los datos y garantiza la integridad y la consistencia de la base de datos.

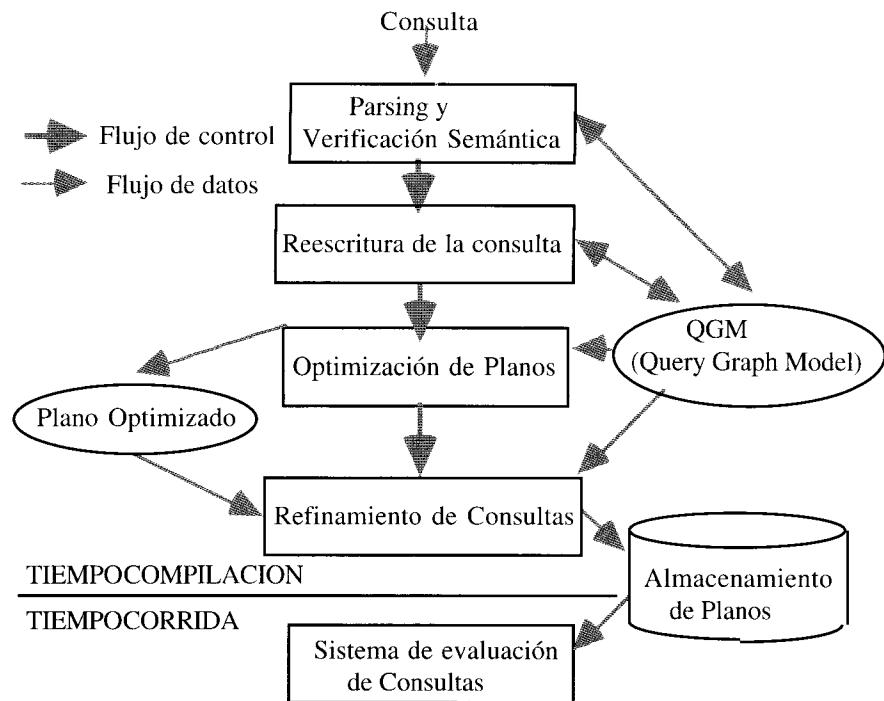


Figura 14.8 Arquitectura de las fases del procesamiento de consultas en Starburst

14.5.3 El sistema Iris

Finalmente se estudia un sistema que puede ser considerado como un híbrido de los dos tipos estudiados hasta ahora. En efecto, es bastante relacional en su estilo y tiene fuertes influencias del Prolog.

Este sistema, conocido como *Iris* fue desarrollado como un sistema experimental por Hewlett-Packard Laboratories en Palo Alto, California.

Iris se sustenta sobre un modelo de datos de tipo funcional, más que un modelo estrictamente orientado a objetos. En la figura 14.9 se presenta la estructura del sistema Iris.



Figura 14.9 Arquitectura del Sistema Iris

El corazón del sistema Iris es el *Administrador de Objetos* -en inglés *Object Manager*-, que no es otra cosa que el procesador de consultas y actualización del sistema. El Administrador de Objetos implementa el Modelo de Datos Iris, el cual se sustenta en los modelos de datos semánticos del tipo Taxis o Daplex [DELO91].

Por su parte, el *Administrador de Almacenamiento* -en inglés *Storage Manager*- es un sub-sistema de almacenamiento relacional clásico y similar al utilizado en el System R, precursor del sistema DB2. Brinda acceso asociativo y posibilidades de actualización a una tabla a la vez, así como un soporte de transacciones.

Con respecto a las interfaces, se tienen tres y son interactivas

- Un driver para la interfaz con el Administrador de Objetos

de tipo funcional, más que totos. En la figura 14.9 se

sp Objects

objetos
es
s. Actualizaciones
ación
s

de la concurrencia
ación
tracción de buffers, índices
uiente

ema Iris

rador de Objetos -en inglés Storage Manager- es un procesador de consultas y gestor de Objetos que implementa el manejo de objetos en los modelos de datos de Iris.

amiento -en inglés Storage Manager- es un sistema relacional clásico y gestor del sistema DB2. Brinda una relación a una tabla a la vez,

s y son interactivas
strador de Objetos

- *Object SQL* -OSQL-, el cual es una ampliación del SQL con capacidades de orientación a objetos.
- Un *Editor de Gráficos* que permite al usuario explorar las estructuras de los metadatos de Iris, así como las estructuras de asociación entre objetos definidos sobre una base de datos Iris.

Iris acepta herencia múltiple, así como el manejo de versiones.

Ejemplo. Si se consideran las clases de objetos que se definieron en el ejemplo anterior, es decir,

TURISTA(Num-Turista, Nom-Turista, País, GUIA(Nom-Guía,Idioma)).

la definición de las mismas, utilizando el lenguaje OSQL, sería:

```
create type Sitio
  (num_sitio      integer required unique,
   nom_sitio      Charstring,
   ciudad         Charstring many);
   pais           Charstring many)

create type Sitio-natural subtype of Sitio
  (tipo            Charstring,
   nombre          Charstring many);
   idioma          Charstring many
   distancia       Charstring many

create type Vestigio_historico subtype of Sitio
  (civilizacion   Charstring,
   período         Charstring);
```

Por su parte, si se desea responder a la siguiente consulta:

Dar los nombre de los sitios ubicados en Costa Rica o Honduras.

Se haría de la siguiente forma:

```
select nom_sitio (g)
for each Sitio_historico g
where pais (g) = 'Costa Rica' or pais (g) = Honduras;
```

14.6 METODOLOGIAS DE ANALISIS Y DISEÑO ORIENTADAS A OBJETOS

En los enfoques clásicos la construcción de un sistema de software debe estar precedido de dos fases bien conocidas. Por un lado el análisis, en donde se determinan las especificaciones del sistema. Por otra parte, se tiene el diseño, en donde se escogen las estrategias que permitan responder a las necesidades de la aplicación.

En el caso del paradigma de orientación a objetos, la diferencia entre el análisis y el diseño se da a nivel de la abstracción. Así, el diseño orientado a objetos a un nivel alto corresponde a una especificación de las clases de objetos y a un nivel más bajo corresponde a la implementación y las decisiones que se deben hacer. Por su parte, en el análisis orientado a objetos no es necesario ningún conocimiento de la implementación [PONC91].

A continuación se van a introducir someramente dos metodologías de análisis orientadas a objetos -OOA de Coad/Yourdon y OOA de Shlaer y Mellor- y dos metodologías de diseño orientadas a objetos -OOD de Booch y HOOD-.

14.6.1 Metodología OOA de Coad/Yourdon

La metodología OOA de Coad y Yourdon es una herramienta de análisis orientado a objetos basada en una serie de 5 etapas. Cada una de las etapas se acompaña de una guía que ayuda parcialmente en la construcción del sistema. Un resumen de esta metodología se presenta en el cuadro 14.1.

Cuadro 14.1 Etapas de la metodología OAA de Coad/Yourdon

1. Definir objetos y clases
2. Definir estructuras
3. Definir áreas sujeto
4. Definir atributos
5. Definir servicios

ISEÑO

de un sistema de software
as. Por un lado el análisis,
el sistema. Por otra parte,
estrategias que permitan

jetos, la diferencia entre el
Así, el diseño orientado a
cificación de las clases de
la implementación y las
análisis orientado a objetos
mentación [PONC91].

mente dos metodologías de
Yourdon y OOA de Shlaer
tadas a objetos -OOD de

n es una herramienta de
de 5 etapas. Cada una de
yuda parcialmente en la
metodología se presenta

de Coad/Yourdon

Definir los objetos y las clases

En esta primera etapa se identifican las clases de objetos, de igual forma como podría hacerse utilizando la metodología introducida en el capítulo 6. Incluso, Coad y Yourdon nos dan una pequeña guía de cómo se podría hacer para determinar las clases de objetos pertinentes a la aplicación que se está modelando.

La definición de las clase de objetos se puede hacer buscando en :

- las estructuras
- los otros sistemas
- los periféricos
- eventos o cosas por memorizar
- papeles jugados
- procedimientos operacionales
- unidades organizacionales.

Definir las estructuras

En esta etapa se identifican dos tipos de estructuras, las herencias y las composiciones. No se da mucha información de cómo pueden interactuar.

Al hacer la definición se buscan las asociaciones entre clases y se representan como

- Generalización / Especialización
- o como
- Compuesto / Componente.

Definir las áreas sujeto

El esquema que representa la aplicación se divide en temas con el fin de establecer subesquemas menos complejos y que sean manejables. Se deben examinar las clases de objetos que son raíz

de una composición y considerarlas como áreas potenciales. Refinar las áreas sujeto para minimizar la interdependencia entre sujetos.

Definir los atributos

En este caso se identifican los atributos que caracterizarán cada una de las clases de objetos, así como las asociaciones que se pueden dar entre las diferentes clases de objetos.

Las características atómicas de los objetos se identifican como atributos de un objeto. También se buscan las posibles asociaciones entre objetos y se determina la cardinalidad de estas asociaciones.

Definir los servicios

En la última etapa de esta metodología se identifican las operaciones que determinarán el comportamiento de los objetos y se establecen los mensajes entre los diferentes objetos.

Para cada clase y objeto, se identifican todos los servicios que realiza, ya sea sobre su propio beneficio o sobre otras clases y objetos.

En la figura 14.10 se muestra el modelaje de una variante del Club de Ecoturismo utilizando la metodología OOA de Coad y Yourdon.

En el esquema de la figura 14.10 los servicios de creación -crea e inicializa un nuevo objeto de una clase-, conexión -conecta o desconecta un objeto con otro objeto-, acceso -da o instala los valores de los atributos de un objeto- y liberación -desconecta y destruye un objeto- de objetos se encuentran implícitos en cada clase de objetos y por lo tanto no se representan en el diagrama.

como áreas potenciales.
a interdependencia entre

que caracterizarán cada una
ciones que se pueden dar

etos se identifican como
buscan las posibles
la cardinalidad de estas

identifican las operaciones
objetos y se establecen los

todos los servicios que
o sobre otras clases y

laje de una variante del
ología OOA de Coad y

ervicios de creación -crea e
-, conexión -conecta o
o -da o instala los valores
desconecta y destruye un
en cada clase de objetos
ama.

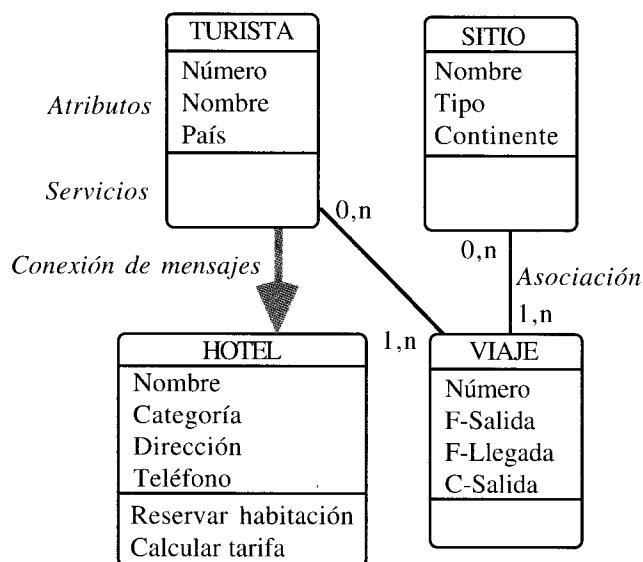


Figura 14.10 Un ejemplo utilizando el OOA de Coad y Yourdon

14.6.2 Metodología OOA de Schlaer y Mellor

Cuando se hace referencia de la metodología OOA de Schlaer y Mellor en realidad se está hablando de tres modelos.

En primer lugar, se tiene un *modelo de información* que permite identificar las entidades conceptuales del mundo real. Las entidades y los enlaces se representan con ayuda del modelo de datos de entidad-asociación.

En segundo lugar se tiene el *modelo de estado* que representa el comportamiento de cada entidad y enlace de esquema de aplicación.

Finalmente, se tiene el *modelo de procesos* que define las acciones asociadas a cada estado del modelo de estado con ayuda de un diagrama de flujo de datos tradicional.

En el cuadro 14.2 se presentan las diferentes etapas de la metodología OOA de Schlaer y Mellor

Cuadro 14.2 Etapas de la metodología OOA de Schlaer y Mellor

1. Desarrollar un modelo de información
2. Definir los ciclos de vida de los objetos
3. Definir las dinámicas de las asociaciones
4. Definir las dinámicas del sistema
5. Desarrollar modelos de procesos
6. Definir dominios y sub-sistemas

A continuación se explican rápidamente las diferentes etapas de la metodología OOA de Schlaer y Mellor.

Desarrollar un modelo de información

El modelo consiste de un conjunto de objetos, atributos, asociaciones y construcción de objetos por medio de las asociaciones *es un* y *es parte de*. Como se dijo anteriormente, en esta metodología un objeto no es otra cosa que una entidad del modelo entidad-asociación.

Definir los ciclos de vida de los objetos

En esta etapa se analiza el ciclo de vida de cada objeto y se formaliza mediante un conjunto de estados, eventos, reglas de transición y acciones. En esta etapa también se definen *timers* que son mecanismos utilizados por las acciones con el fin de generar un evento futuro. En esta etapa se utiliza el modelo de estado introducido anteriormente.

Definir las dinámicas de las asociaciones

En esta etapa se desarrolla un modelo de estado para aquellas asociaciones entre objetos que evolucionan en el tiempo.

Para cada asociación dinámica se define, en el modelo de información, un objeto asociativo.

e Schlaer y Mellor

información
os objetos
asociaciones
ema
os
nas

as diferentes etapas de la

tos, atributos, asociaciones
ociaciones es un y es parte
ología un objeto no es otra
ciación.

cada objeto y se formaliza
s, reglas de transición y
mers que son mecanismos
ar un evento futuro. En esta
ido anteriormente.

o de estado para aquellas
en el tiempo.

define, en el modelo de

Definir las dinámicas del sistema

Esta etapa produce un modelo de tiempo y control a nivel del sistema.

Desarrollar modelos de procesos

Para cada acción, se crea un diagrama de flujo de datos que muestra los procesos de tal acción, y los flujos de los datos a través de los procesos.

Definir dominios y sub-sistemas

Para problemas grandes y complejos, puede ser útil descomponer el problema en dominios conceptualmente distintos. Se identifican cuatro tipos de dominios: de aplicación, de servicio, de arquitectura y de implementación.

14.6.3 Metodología OOD de G. Booch

Possiblemente la metodología de Booch es la herramienta de diseño orientado a objetos más antigua [BOOC86] la cual se sustenta en características del lenguaje de programación ADA bajo un estilo orientado a objetos.

Cuadro 14.3 Etapas de la metodología OOD de Booch

1. Identificar clases y objetos
2. Identificar la semántica de las clases y objetos
3. Identificar asociaciones entre clases y objetos
4. Implementar clases y objetos

A continuación se explican someramente estas cuatro etapas [BOOC91].

Identificar clases y objetos

La primera etapa busca identificar las entidades fundamentales del sistema, es decir las clases y los objetos relevantes así como definir los

mecanismos que establezcan el comportamiento requerido de los objetos que trabajan juntos con el fin de realizar cierta función.

Identificar la semántica de las clases y objetos

Esta etapa consiste en establecer los significados de las clases y los objetos identificados en la etapa anterior. Una técnica que puede ser útil para realizar esta actividad es escribir un escenario que defina el ciclo de vida de cada objeto desde su creación hasta su destrucción, incluyendo sus comportamientos característicos.

Identificar asociaciones entre clases y objetos

Se establecen las interacciones de las clases y objetos, tales como patrones de herencia entre clases y patrones de cooperación entre objetos. Así, en esta etapa se deben descubrir modelos entre clases que permitan reorganizar y simplificar las estructuras de las clases y modelos entre conjuntos cooperativos de objetos. Además, se deben tomar decisiones de cómo las clases se ven entre ellas, cómo los objetos se ven entre ellos y cuáles objetos y clases no deben verse entre ellos.

Implementar clases y objetos

En esta etapa se toman las decisiones de diseño referente a la representación de las clases y objetos determinadas anteriormente y asignar estas clases y objetos en los diferentes módulos y los programas en los procesadores.

14.6.4 Metodología HOOD

La metodología *HOOD* -Hierarchical Object Oriented Design- es una herramienta de diseño jerarquizado para el desarrollo de software complejo, científico y de tiempo real en el lenguaje ADA. HOOD fue desarrollado por la Agencia Espacial Europea y se sustenta en la metodología *GOOD* -General Object Oriented Design- que fue a su vez desarrollado por la NASA en 1986 [GRAH91].

o requerido de los objetos para su función.

os

cados de las clases y los métodos. Una técnica que puede ser útil es el marco que define el ciclo de vida y la destrucción, incluyendo

s

es y objetos, tales como los mecanismos de cooperación entre los modelos entre clases que se crean. Los tipos de las clases y modelos que se crean, además, se deben tomar en cuenta para ver cómo los objetos se ven interactuando entre ellos.

el diseño referente a la programación minadas anteriormente y los módulos y los programas

Oriented Design- es una forma de desarrollo de software en el lenguaje ADA. HOOD fue desarrollada y se sustenta en la metodología de Design- que fue a su vez

HOOD se inspira de los mecanismos de diseño propuestos por G. Booch [BOOC86] y los formaliza por medio de una representación gráfica y de una descripción textual. En la figura 14.11 se muestra la representación gráfica de los objetos en HOOD.

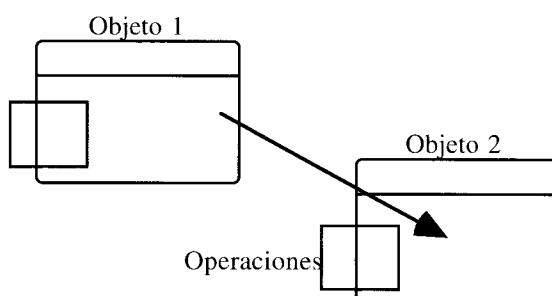


Figura 14.11 Representación gráfica de los objetos en HOOD

Cada objeto en HOOD se formaliza por medio de una representación gráfica y una notación textual. La descripción textual sirve de lenguaje de descripción de programas en la fase de diseño y se hace por medio de un *esqueleto de definición de objetos* -del inglés ODS: Object Description Skeleton- en donde cada campo se describe por medio de un lenguaje HOOD_PDL o ADA_PDL [PONC91].

El aspecto estructural del objeto se describe por medio de una interfaz y una parte que contiene el cuerpo -algoritmo de descripción- del objeto.

Por su parte, el comportamiento se describe por medio de las operaciones, las cuales son activadas por medio de los flujos de control. Un *flujo de control* corresponde a la ejecución de una operación que puede activarse en forma *secuencial* -se transfiere el control del usuario al objeto utilizado y la operación se efectúa inmediatamente según la estructura de control de la operación asociada- o en forma *paralela* -el control se transfiere según un protocolo que depende tanto del estado interno del objeto como de la interacción de los flujos de control que entran en el objeto.

Con respecto a las asociaciones entre los objetos se puede hablar de dos grandes tipos.

Por una parte se tiene la *asociación de inclusión* -del inglés include relationship- en donde un objeto se descompone por medio de etapas elementales, en un conjunto de objetos hijos de más bajo nivel.

Por otra parte se tiene la *asociación de utilización* -del inglés use relationship- la cual define una jerarquía entre los objetos -entre dos objetos solo existe una flecha que representa esta asociación de utilización-. Un objeto utiliza otro objeto si el primero requiere de una o más operaciones producidas por el segundo objeto.

Asimismo, se pueden mencionar, entre otras, las siguientes metodologías:

- *OMT* de Rumbaugh [RUMB91]
- *Fusion* de Coleman [COLE94]
- *RDD* de Wirfs y Brock [WIRF90]
- *Objectory* de I. Jacobson [SACO92]

Finalmente, se puede también mencionar el *Unified Method* de G. Booch y J. Rumbaugh [BOOC95], que representa la unificación de las metodologías OMT y Booch y persigue la presentación de un estándar en el dominio del análisis y diseño orientados a objetos.

EJERCICIOS Y PREGUNTAS DE REPASO

14.1 Definir los siguientes conceptos:

- objeto
- clase de objetos
- abstracción
- encapsulamiento
- persistencia

objetos se puede hablar de

clusión -del inglés include
pone por medio de etapas
de más bajo nivel.

utilización -del inglés use
entre los objetos -entre dos
senta esta asociación de
primero requiere de una o
bjeto.

re otras, las siguientes

91]

RF90]

92]

el *Unified Method* de G.
senta la unificación de las
representación de un estándar
s a objetos.

14.2 Establecer los diferentes criterios que debe cumplir un SABD para ser considerado orientado a objetos.

14.3 Dar una lista de tres SABDOOs y explicar someramente sus arquitecturas.

14.4 Sea el esquema de la figura 14.10. Responder a la siguiente consulta utilizando el lenguaje CO₂ del sistema O₂ y el lenguaje OPAL del sistema GemStone:

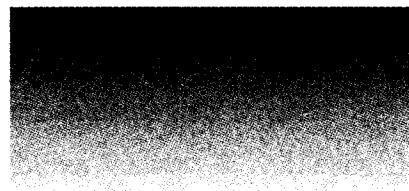
*Seleccionar los turistas que son de México y que tienen una
reservación en el Hotel Emporio entre el 15 y el 21 de junio de
1996.*

14.5 Utilizando el OOA de Coad y Yourdon modelar los problemas presentados en el capítulo 6.

14.6 Explicar en qué consiste la metodología OOA de Schlaer y Mellor.

14.7 Explicar las cuatro etapas de la metodología OOD de Booch.

14.8 Describir cómo se representa un objeto en la metodología HOOD.



15

Otras Tendencias

«La computadora ha desencadenado un alud de ideas nuevas sobre el hombre como parte interactiva de sistemas más amplios, sobre su fisiología, su manera de aprender, su manera de tomar decisiones. Virtualmente, toda disciplina intelectual, desde la ciencia política hasta la psicología familiar, ha sufrido el impacto de una serie de hipótesis imaginativas, provocadas por el invento y la difusión de la computadora... aunque el impacto mayor no se ha producido aún. Así se acelera el ciclo innovador alimentándose de sí mismo.»

Alvin Toffler,
periodista estadounidense.

15. 1 INTRODUCCION

En los primeros trece capítulos de la presente obra se ha hecho un recorrido general de la tecnología de bases de datos relacionales.

Esta tecnología es madura y contabiliza cientos de millones de dólares al año.

Sin embargo, en los últimos años ha hecho aparición un nuevo tipo de aplicaciones y estos sistemas se muestran muy limitados para responder a estas nuevas tendencias.

Una de las estrategias que se ha seguido, como se estudió en el capítulo anterior, es la incorporación del paradigma de la orientación a objetos a los sistemas de bases de datos.

Aun así, quedan pendientes otras nociones en donde la orientación a objetos no permite plenamente atacar los problemas que se perciben.

De esta forma, debido a la necesidad de contar con sistemas de información que reflejen más fielmente el mundo real, se hace necesario considerar una serie de parámetros que han sido dejados de lado, más por limitaciones en la capacidad del hardware que por limitaciones del software. Entre estos parámetros se tienen los referentes a la temporalidad, la deducción, la incertidumbre y el manejo de diferentes dispositivos no tradicionales.

Así por ejemplo, el médico que desea contar con herramientas que le permitan manejar datos, que muchas veces son incompletos o inciertos, difícilmente lo podría hacer con un sistema relacional tradicional. Es el caso del diagnóstico de algún tipo de enfermedad, en donde, con base en información imprecisa por parte del paciente, éste debe diagnosticar alguna dolencia. Al médico también le interesaría contar con un historial del paciente en un computador y quisiera saber su condición en determinada época. Además, podría ser muy útil contar con las radiografías del paciente guardadas en algún dispositivo de almacenamiento conectado a un computador.

Este tipo de necesidades se pueden encontrar casi en cualquier actividad humana, entre otras, la economía, la toma de decisiones, el diseño por computador, la educación, etc.

Otro ejemplo es el referente al estudio del ADN o Ácido desoxirribonucleico que es una macromolécula que se encuentra en el núcleo de cada célula de cualquier ser viviente. Este tipo de estudio por computador requerirá otra clase de sistemas de bases de datos.

En efecto, el ADN es el responsable de la transmisión de las características hereditarias -o código genético-. La información contenida en los genes se puede ver como un conjunto de órdenes o instrucciones con el fin de producir nuevos genes iguales a los originales. El material genético es uno de los más estables que se conocen; de otra forma sería inútil en su función de acarrear información hereditaria. Sin embargo, se pueden dar cambios estructurales en los genes pues éstos se encuentran expuestos a factores físicos y químicos que pueden alterar dicho código, cuando estos cambios se dan, se produce lo que se conoce como *mutación*. Así, cuando el código se altera por medio de la mutación, se pueden desencadenar enfermedades graves como el cáncer.

Para buscar formas de curar el cáncer están los mecanismos que permitan la comparación de los códigos genéticos de células normales con células cancerosas. Sin embargo, estas comparaciones no son evidentes, pues el código genético requiere de miles de millones de caracteres para ser manejados y almacenados en un computador. Los SABDs actuales no son suficientes para resolver el problema de este tipo de bases de datos científicas, pues se requiere la administración de objetos complejos, así como la capacidad de definir reglas que permitan deducir nueva información a partir de los datos que se encuentran almacenados en la base de datos.

Una conjunción del paradigma de los sistemas de bases de datos con los sistemas basados en conocimiento, conocidos como *SABDs deductivos*, pueden ser una alternativa real para diseñar sistemas que permitan atacar estos problemas.

contrar casi en cualquier
, la toma de decisiones, el

udio del ADN o Ácido
ula que se encuentra en el
nte. Este tipo de estudio por
de bases de datos.

de la transmisión de las
enético-. La información
un conjunto de órdenes o
enes iguales a los originales.
les que se conocen; de otra
información hereditaria. Sin
s en los genes pues éstos se
uímicos que pueden alterar
se produce lo que se conoce
e altera por medio de la
ades graves como el cáncer.
están los mecanismos que
néticos de células normales
as comparaciones no son
e de miles de millones de
los en un computador. Los
ver el problema de este tipo
uiere la administración de
definir reglas que permitan
s datos que se encuentran

emas de bases de datos con
conocidos como *SABDs*
para diseñar sistemas que

Cuando además se adiciona a estos sistemas el paradigma de orientación a objetos, éstos se conocen como *SABDs deductivos orientados a objetos* -en inglés *Deductive and Object-Oriented Databases (DOODB)*-.

Asimismo cuando se incorpora el concepto de incertidumbre a los sistemas de bases de datos, los mismos se pueden llamar *difusos* -en inglés *fuzzy*-, y cuando se incorpora el tiempo los sistemas se denominan *temporales*.

Finalmente, la necesidad de manejar una base de datos desde diferentes dispositivos y fuentes de información -texto, imágenes, sonido, etc.-, se denominan sistemas de *multimedios*.

Así, se ha considerado interesante terminar el presente libro con una introducción muy general a estos conceptos, los cuales formarán parte de la tecnología de bases de datos en el próximo milenio.

15. 2 BASES DE DATOS DE MULTIMEDIOS

En la actualidad, gracias al desarrollo tan importante que se ha dado en el campo de la electrónica, es posible integrar diferentes dispositivos, como por ejemplo, videograbadoras, computadores, discos ópticos, etc.

De ahí y por la necesidad de contar con información más fiel, nace la idea de concebir sistemas que permitan el almacenamiento y manejo de diversos tipos de datos, generados por estos diferentes dispositivos. Dichos datos -por ejemplo, imágenes, sonidos, videos, mapas, textos, etc.-, se denominan *multimedios*.

Entre las posibles aplicaciones de un sistema basado en datos multimedios, se pueden dar las siguientes:

- Archivos de lectura, como por ejemplo, diarios electrónicos, imágenes médicas, enciclopedias, etc.
- Ayuda en el proceso enseñanza-aprendizaje, juegos y otras ayudas didácticas.

- Ingeniería de diseño, publicidad y otras aplicaciones similares.
- Control de tráfico, control de procesos, etc.

A continuación se establecen cuáles son los componentes básicos que permiten concebir una base de datos de este tipo.

Dispositivos de entrada y salida

Con respecto a los dispositivos de entrada y salida, se pueden considerar los tradicionales como el teclado, el mouse y el monitor. Sin embargo, en el mundo de los multimedios existen otros componentes, como son el *escáner*, que permite capturar una imagen y convertirla en un archivo de computador de tipo raster. También, se puede hablar de la *cámara de video*, la cual genera una señal analógica y se graba en un VCR -*video cassette recorder*-. Asimismo, los micrófonos y parlantes para grabar y emitir sonidos.

Dispositivos de almacenamiento

Los datos multimedios requieren de dispositivos de gran capacidad de almacenamiento. Así por ejemplo, una imagen con sonido de varios minutos requiere varios megabytes. Grabar varios minutos de video, resulta aun más costoso.

Con tales cantidades de memoria, los medios tradicionales son bastantes limitados.

Así surgen los llamados *discos ópticos*. Entre estos discos, se pueden mencionar el *CD-ROM* -Compact Disc Read Only Memory-, el *WORM* -Write Once, Read Many times- y los *discos borrables*.

El VCR es un dispositivo de almacenamiento de video relativamente barato.

Un SABD que permita el manejo de datos de multimedios, debe incluir mecanismos para la edición de los diferentes datos, así como un manejo eficiente de los diferentes dispositivos de entrada y salida.

plicaciones similares.

tc.

componentes básicos que
po.

da y salida, se pueden
l mouse y el monitor. Sin
isten otros componentes,
a imagen y convertirla en
ién, se puede hablar de la
alógica y se graba en un
s micrófonos y parlantes

ivos de gran capacidad de
en con sonido de varios
varios minutos de video,

medios tradicionales son

re estos discos, se pueden
Only Memory-, el WORM
orables.

to de video relativamente

os de multimedios, debe
rentes datos, así como un
de entrada y salida.

Una de las consideraciones que deben tenerse en cuenta es la referente a lo que Meyer-Wegener denomina *independencia del dispositivo* [MEYE92]. Esto significa que, si se desea trasladar datos de un dispositivo a otro más eficiente, se debería minimizar el problema de este traslado.

Otro hecho importante es la *independencia del formato*, es decir, que una aplicación puede requerir diferentes tipos de datos en formatos diferentes. Así, el sistema debería suprir a cada aplicación el formato que requiera, mientras se esconde el formato de almacenamiento interno real.

Otra idea importante en el contexto de un SABD de multimedios, es que debe permitir la asociación que se puede dar entre diferentes tipos de datos. En un ambiente de multimedios se pueden tener las siguientes [MEYE92]:

- *Objeto-atributo*: los objetos -entidades- como personas, autos, etc., pueden describirse por medio de fotos, gráficos, o sonidos. Así, un objeto en este ambiente muestra una propiedad del objeto representado en el sistema.
- *Composición*: es el caso de objetos de un tipo, compuestos por objetos de otros medios. Por ejemplo, un documento puede asociarse con un texto, una fotografía, un gráfico, etc.
- *Equivalencia*: es cuando un objeto de multimedios puede representarse en diferentes medios. Así, el sistema debe permitir la liga entre diferentes medios y manejar la equivalencia de dichos objetos.

Otro problema que debe atacarse es el referente a la búsqueda de los datos de multimedios y las posibles relaciones entre ellos, por medio de consultas de usuario. Por esta razón deben establecerse, álgebras de comparación entre los diferentes objetos.

Finalmente, una idea interesante que debe mencionarse es la de rescatar el paradigma inicial del modelo relacional y en particular, lo referente a los dominios de las diferentes tablas. En efecto, si se amplía

el concepto de dominio y deja de ser un tipo de dato definido por el sistema para ser definido por los usuarios, entonces sería posible establecer relaciones en un ambiente de multimedios, como el que se presenta en la figura 15.1 [DARW95].

AVE

Nombre	Descripción	Cuadro	Video	Sonido	Migración
Carpintero					
Jilguero					

where	Nombre	es el nombre de un ave	[texto]
and	Descripción	describe el ave	[texto]
and	Cuadro	es una ilustración del ave	[foto]
and	Video	es un film del ave	[video]
and	Sonido	es el trino del ave	[audio]
and	Migración	es la ruta de migración del ave	[mapa]

Figura 15.1 Ejemplo de una relación con multimedios.

En este ejemplo, se define una relación sobre las aves de una determinada región. En este caso, se establecen dominios como el audio, el cual permite incluir un atributo sonido, que brinda información sobre el trino de las diferentes aves concernidas.

15.3 BASES DE DATOS DEDUCTIVAS

Una *base de datos deductiva* debe contar al menos con las siguientes características [SHAL91]:

- Tener la capacidad de expresar consultas por medio de reglas lógicas.
- Permitir consultas recursivas y algoritmos eficientes para su evaluación.
- Contar con negaciones estratificadas.
- Soportar objetos y conjuntos complejos.

o de dato definido por el s, entonces sería posible ltimedios, como el que se

Sonido	Migración

ave [texto]
el ave [foto]
[video]
[audio]
ción del ave [mapa]

n multimedios.

n sobre las aves de una n dominios como el audio, brinda información sobre

l menos con las siguientes

tas por medio de reglas

tmos eficientes para su

- Contar con métodos de optimización que garanticen la traducción de especificaciones dentro de planes eficientes de acceso.

Para comprender mejor el concepto de bases de datos deductivas, es importante tomar en cuenta lo expuesto en el capítulo 3 sobre el modelo relacional de Codd.

En efecto, se mencionó que una relación $R(A_1, A_2, \dots, A_n)$ de una base de datos se puede ver como un predicado $P(x_1, x_2, \dots, x_n)$ de la lógica de primer orden de tal forma que una tupla $t = [a_1, a_2, \dots, a_n]$ pertenece a dicha relación si $P(a_1, a_2, \dots, a_n)$ es verdadero.

Como una de las características fundamentales de una base de datos deductiva es la posibilidad de inferir información a partir de los datos almacenados, es imperativo modelar la base de datos como un conjunto de fórmulas lógicas, las cuales permiten inferir otras fórmulas nuevas.

Antes de introducir formalmente el concepto de base de datos deductiva se hace necesario recordar varios conceptos que fueron introducidos en el capítulo 3.

En primer lugar, un *término* se define recursivamente de la siguiente forma:

1. Toda constante es un término
2. Toda variable es un término
3. Si f es un símbolo de función n -aria y t_1, t_2, \dots, t_n son términos, entonces $f(t_1, t_2, \dots, t_n)$ es un término
4. Todos los términos se generan sólo usando las reglas anteriores

Por otra parte, un *átomo* es cualquier expresión de la forma:

$$P(t_1, t_2, \dots, t_n)$$

en donde P es un predicado y cada t_i representa un término.

Un átomo se llama también *literal positiva* y la negación del mismo se conoce como *literal negativa*, es decir, respectivamente las siguientes expresiones:

$$P(t_1, t_2, \dots, t_n) \text{ o bien } \neg P(t_1, t_2, \dots, t_n)$$

Con lo anterior se establece el concepto de cláusula de Horn. Así, una *cláusula de Horn* es una fórmula en la lógica de primer orden que se puede escribir de la siguiente forma:

$$A \vee \neg B_1 \vee \dots \vee \neg B_n$$

En este caso A y cada B_i representan una literal.

Según esta definición, una cláusula de Horn, por las propiedades de las conectivas lógicas, es equivalente a la siguiente fórmula:

$$B_1 \wedge \dots \wedge B_n \Rightarrow A$$

la cual será denotada de la siguiente forma:

$$A \Leftarrow B_1 \wedge \dots \wedge B_n$$

y se llama también *cláusula de base de datos*.

Con estas definiciones, se puede decir que una *base de datos deductiva* es una base de datos relacional junto un conjunto de cláusulas de bases de datos.

Ejemplo. En la figura 15.2 se muestran algunos distritos y cantones de las provincias de San José y Alajuela.

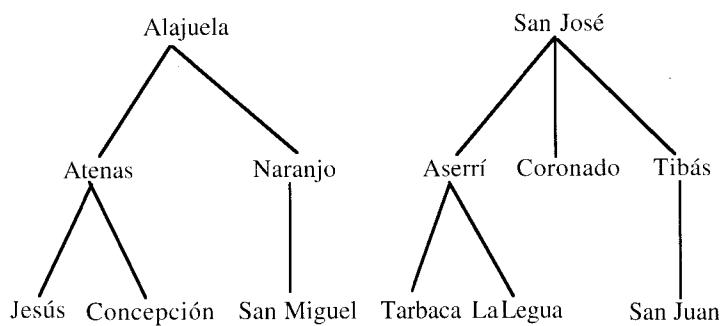


Figura 15.2 Ejemplo de distritos y cantones de Costa Rica

la negación del mismo
ivamente las siguientes

$\dots t_n$)

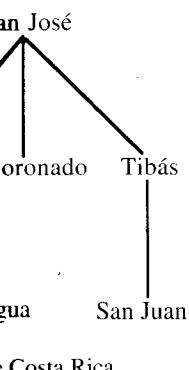
usula de Horn. Así, una
e primer orden que se

al.

por las propiedades de
te fórmula:

e una base de datos
n conjunto de cláusulas

istritos y cantones de las



Bajo este contexto, y tomando en cuenta las definiciones anteriores, se va a definir una base de datos deductiva.

Base de datos relacional

Los valores de la base de datos se pueden representar de la siguiente forma:

LUGAR(x): x es un lugar	ES-DISTRITO(x,y): x es distrito de y ES-CANTON-DE(x,y) : x es cantón de y
LUGAR(Concepción)	ES-DISTRITO-DE(Concepción, Atenas)
LUGAR(Jesús)	ES-DISTRITO-DE(Jesús, Atenas)
LUGAR(La Legua)	ES-DISTRITO-DE(San Miguel, Naranjo)
LUGAR(San Juan)	ES-DISTRITO-DE(La Legua, Aserrí)
LUGAR(San Miguel)	ES-DISTRITO-DE(Tarbaca, Aserrí)
LUGAR(Tarbaca)	ES-DISTRITO-DE(San Juan, Tibás)
LUGAR(Aserrí)	ES-CANTON-DE(Atenas, Alajuela)
LUGAR(Atenas)	ES-CANTON-DE(Naranjo, Alajuela)
LUGAR(Coronado)	ES-CANTON-DE(Coronado, San José)
LUGAR(Naranjo)	ES-CANTON-DE(Aserrí, San José)
LUGAR(Tibás)	ES-CANTON-DE(Tibás, San José)
LUGAR(Alajuela)	
LUGAR(San José)	

Cláusulas de base de datos

1. Regla de integridad: Cada distrito, cantón o provincia es un lugar, se puede representar de la siguiente forma:

$$\begin{aligned} \text{LUGAR}(x) &\Leftarrow \text{ES-DISTRITO-DE}(x,y) \\ \text{LUGAR}(y) &\Leftarrow \text{ES-DISTRITO-DE}(x,y) \end{aligned}$$

$$\begin{aligned} \text{LUGAR}(x) &\Leftarrow \text{ES-CANTON-DE}(x,y) \\ \text{LUGAR}(y) &\Leftarrow \text{ES-CANTON-DE}(x,y) \end{aligned}$$

2. Regla de deducción: La relación SE-ENCUENTRA-EN(x,y) : el distrito x se encuentra ubicado en la provincia y se puede presentar de la siguiente forma:

$$\text{SE-ENCUENTRA-EN}(x,y) \Leftarrow \text{ES-DISTRITO-DE}(x,z) \wedge \text{ES-CANTON-DE}(z,y)$$

Esto significa que si el distrito x pertenece al cantón z y z es un cantón de la provincia y entonces x es un lugar que se ubica en la provincia y.

Según la base de datos inicial, entonces se pueden deducir los siguientes hechos

SE-ENCUENTRA-EN(Concepción, Alajuela)
SE-ENCUENTRA-EN(Jesús, Alajuela)
SE-ENCUENTRA-EN(La Legua, San José)
SE-ENCUENTRA-EN(San Juan, San José)
SE-ENCUENTRA-EN(San Miguel, Alajuela)
SE-ENCUENTRA-EN(Tarbaca, San José))

Por su parte, bajo este enfoque una *consulta* se puede expresar de la forma:

$$\Leftarrow B_1 \wedge \dots \wedge B_n$$

con x_1, x_2, \dots, x_m variables libres involucradas en las literales.

La respuesta a dicha consulta consiste de los $[a_1, a_2, \dots, a_n]$, tal que al reemplazarlos por las variables, utilizando las cláusulas de bases de datos definidas previamente, la fórmula se hace válida.

Ejemplo. Siguiendo con el ejemplo anterior, la respuesta a la consulta:

Dar la lista de los distritos de los diferentes cantones de Alajuela
se representaría de la siguiente forma:

$$\Leftarrow \text{ES-CANTON-DE}(x, \text{Alajuela})$$

La respuesta es entonces el conjunto de valores x que hace el predicado válido. En este caso sería lo siguiente:

ES-CANTON-DE(Atenas, Alajuela)
ES-CANTON-DE(Naranjo, Alajuela)

Un lenguaje de consultas que ha sido estudiado en el marco de las bases de datos deductivas es el DATALOG, que se basa en cláusulas de Horn y se puede ver como una variación del lenguaje de programación Prolog.

Finalmente, en la figura 15.2 se muestra la arquitectura de un SABD deductivo con sus dos componentes fundamentales: un módulo deductivo y un SABD relacional.

ducir los siguientes hechos

se puede expresar de la

en las literales.

$[a_1, a_2, \dots, a_n]$, tal que al
s cláusulas de bases de
válida.

esta a la consulta:
de Alajuela

hace el predicado válido. En

ado en el marco de las
se basa en cláusulas de
enguaje de programación

quitectura de un SABD
mentales: un módulo

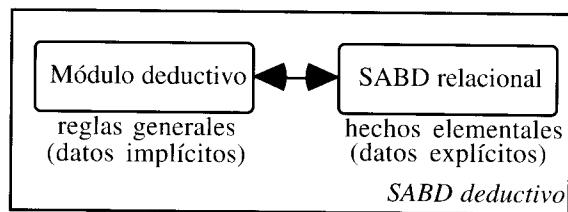


Figura 15.2 Arquitectura de un SABD deductivo

15.4 BASES DE DATOS DIFUSAS

Uno de los problemas más importantes que actualmente interesan a muchos investigadores es el referente al manejo de la incertidumbre y la vaguedad en la información. Para atacar este problema se han sugerido modelos basados en la teoría de las probabilidades, teoría de la certeza, teoría de la evidencia, y los conjuntos difusos.

El conjunto difuso es un concepto introducido por Lotfi Zadeh en 1965 [ZADE65] y que descansa en las siguientes consideraciones.

Si se considera a X un conjunto de elementos en el sentido tradicional de la teoría de conjuntos, se puede definir un conjunto difuso $X\sim$ sobre X , como la función dada a continuación:

$$X\sim = \{ (x, \mu(x)) / x \in X \text{ y } \mu: X \rightarrow [0,1] \}$$

La función $\mu: X \rightarrow [0,1]$, se denomina *función de pertenencia*.

Es interesante mencionar que, bajo este concepto, un conjunto en el sentido tradicional es un caso particular de un conjunto difuso. En efecto, si $\mu(x) = 0$, se dice que el elemento x no está en el conjunto. Si $\mu(x) = 1$ se dice que el elemento x pertenece plenamente al conjunto.

Con ayuda de los conjuntos difusos, muchos conceptos lingüísticos como *grande*, *poco* y *bastante*, podrían ser representados y manejados apropiadamente.

Ejemplo. En la figura 15.3, se presenta un conjunto difuso que muestra el concepto de estatura de un grupo de personas. Bajo este contexto, una persona con una estatura menor de 150 cm, puede considerarse como no alto o pequeño.

Por su parte, una persona con una estatura de más de 190 cm puede considerarse alta y por lo tanto elemento plenos del conjunto.

Si se considera $X = \{\text{Pedro, Pablo, Jorge, Juan}\}$ el conjunto de jugadores de un equipo de baloncesto, sobre el concepto alto se puede definir el siguiente conjunto difuso $X_{\sim} = \{(\text{Pedro}, 0.2), (\text{Pablo}, 0.5), (\text{Jorge}, 0.55), (\text{Juan}, 0.9) \}$. A partir de este conjunto es posible deducir algunos hechos: por ejemplo: Pedro es el más pequeño del grupo; Pablo y Jorge tienen aproximadamente la misma estatura y Juan es el más alto del grupo, pero no muy alto.

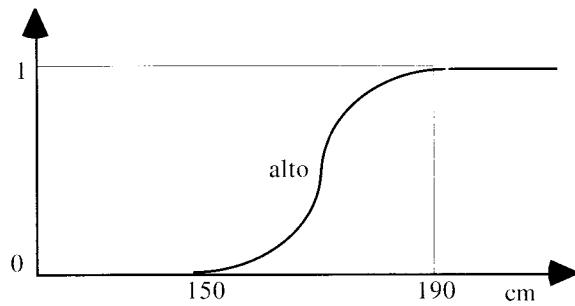


Figura 15.3 Ejemplo de un conjunto difuso

A partir de un conjunto difuso, se puede construir toda una teoría, como se hace en el caso tradicional. En efecto, se pueden definir relaciones difusas, grafos difusos, análisis difuso, etc.

Para el caso que interesa en esta sección, se define a continuación una relación difusa.

Sean X y Y dos subconjuntos de U -considerando el conjunto universal-. Se dice que el conjunto:

$$R = \{ (x,y; \mu_R(x,y)) / (x,y) \in X \times Y \}$$

es una *relación binaria difusa* sobre $X \times Y$ [ZIMM86], en donde

$$\mu_R(x,y): X \times Y \rightarrow [0,1]$$

es la función de pertenencia.

o difuso que muestra el concepto en contexto. una persona con una altura no alto o pequeño.

de 190 cm puede considerarse

el conjunto de jugadores de un equipo que define el siguiente conjunto

(Juan, 0.9) }. A partir de este ejemplo: Pedro es el más pequeño que la misma estatura y Juan es el

190 cm

dato difuso

para construir toda una teoría, efecto, se pueden definir uso, etc.

que define a continuación una

-considerado el conjunto

$\in X \times Y \}$

[ZIMM86], en donde

1]

En el caso en que X y Y son conjuntos finitos, tal relación se puede representar por medio de una matriz.

Ejemplo. Sean $X = Y = \{\text{Ana, Juan, Carla, María}\}$. Sea R : “ x es mucho mayor que y ”, se puede representar esta situación según se aprecia en la figura 15.4. En este caso, se puede ver que la más joven es María y que Carla es la más vieja, etc.

	Ana	Juan	Carla	María
Ana	0	0.3	0	0.7
Juan	0	0	0	0.2
Carla	0.5	0.7	0	1
María	0	0	0	0

Figura 15.4 Ejemplo de una relación binaria difusa

Con respecto a un sistema de bases de datos, se han hecho esfuerzos para incorporar lo difuso, sobre todo a nivel relacional. Estos esfuerzos se pueden clasificar en tres grandes áreas:

- Datos no difusos y consultas difusas
- Datos difusos y consultas difusas
- Datos y consultas difusas.

Ejemplo. En la figura 15.5 se aprecia lo que puede ser una relación con datos difusos. Así, la edad de Carlos se encuentra entre los 20 y 30 años, gana alrededor de 50000 colones por mes y hay bastante certeza de que esté casado.

EMPLEADO

Nombre	Estado Civil	Edad	Salario
Carlos	{(01.,D), (0.9,C)}	[20,30]	Aproximadamente 50000
Juan	no aplicable	45	76000
Pedro	S	desconocido	{45000, 50000}

Figura 15.5 Ejemplo de una relación difusa en una base de datos

Por su parte se podría utilizar un lenguaje que permita consultas difusas, es decir, que se encuentren más cercanas al lenguaje natural. Si se usara una versión difusa

del lenguaje de consultas Quel del sistema Ingres [CORD95], la consulta: *dar una lista de las personas que son jóvenes y que ganan muy poco*, se podría representar de la siguiente manera:

```
retrieve nombre  
from empleado  
where salario ≈muy_poco and edad ≈joven
```

En la literatura se pueden encontrar varias propuestas para ampliar el modelo relacional para el manejo de datos difusos.

Una de las ideas que deben desarrollarse es la referente a los operadores relacionales, los cuales deben ampliarse, con el fin de manipular datos difusos.

En este contexto, el modelo de Buckles y Petry [BUCK82a] introduce el concepto de relación de similitud, concepto inicialmente definido por L. Zadeh. La idea de la similitud radica en el hecho de que las relaciones de equivalencia o identidad entre valores de dominio, son reemplazadas por una medida de cercanía, que es precisamente la relación de similitud. A continuación se discute con más detalle esta propuesta.

En una base de datos relacional tradicional, cada tupla de una relación se define como un arreglo $[d_1, d_2, \dots, d_n]$ en donde cada d_i representa un valor atómico tomado de un dominio. Por su parte, en una base de datos relacional difusa cada d_i puede ser un conjunto de valores diferente del conjunto vacío.

En segundo lugar, cada atributo A tiene asociado una *matriz de similitud* S definida sobre el dominio D de A de la siguiente manera:

$$S: D \times D \rightarrow [0,1]$$

Una relación de similitud es una generalización del concepto de relación de equivalencia en que si $x, y, z \in D$, entonces se tiene que S es:

$$\text{reflexiva: } S(x,x) = 1$$

ORD95]. la consulta: *dar una
y poco.* se podría representar

ropuestas para ampliar el
sos.

e es la referente a los
mpliarse, con el fin de

ry [BUCK82a] introduce
nicialmente definido por
cho de que las relaciones
minio, son reemplazadas
e la relación de similitud.
propuesta.

ada tupla de una relación
de cada d_i representa un
nte, en una base de datos
de valores diferente del

asociado una matriz de
la siguiente manera:

del concepto de relación
tiene que S es:

simétrica: $S(x,y) = S(y,x)$

transitiva: $S(x,z) \geq \max_{y \in D} [\min S(x,y) S(y,z)]$

Así, en el contexto el modelo de Buckles y Petry, una relación difusa R se define como un subconjunto del producto cartesiano

$$P(D_1) \times P(D_2) \times \dots \times P(D_n)$$

en donde $P(X)$ representa el conjunto de subconjuntos de X. Bajo estas circunstancias una tupla difusa se define de la siguiente forma:

$$t = [d_1, d_2, \dots, d_n]$$

en donde cada d_i es un subconjunto de su dominio respectivo.

Por su parte, una *interpretación* de una tupla $t = [d_1, d_2, \dots, d_n]$ es cualquier asignación de valores $[d_1, d_2, \dots, d_n]$ tal que cada d_i es un elemento de su dominio respectivo.

Por su parte, dado un atributo A sobre un dominio D, se define el umbral de similitud de A de la siguiente forma

$$\text{umb}(A) = \min [\min S(x,y)]$$

$\forall i \quad x, y \text{ recorriendo la columna de } A$

Es interesante mencionar que en el caso de una relación no difusa, se tiene que el umbral del atributo es 1.

Existen otras propuestas como el modelo de Kacprzyk y Ziolkowski [KACP86], y se basa en expresiones difusas. En este modelo, el almacenamiento no es difuso, lo que persigue es la recuperación de los datos por medio de consultas difusas, como por ejemplo, *encontrar los registros en los cuales el atributo A es largo.*

La estrategia es calcular primero el grado de pertenencia para una tupla dada y si éste supera un umbral dado, entonces la tupla es parte de la respuesta.

15.5 BASES DE DATOS TEMPORALES

Un concepto que ha quedado un poco al margen de los sistemas tradicionales de bases de datos es el referente al tiempo. Sin embargo, este concepto es fundamental para comprender cómo evoluciona el mundo real, del cual una base de datos quiere ser una representación más o menos cercana. Una base de datos actual representa un instante en la vida de un sistema. En muchas aplicaciones, los datos pueden depender del tiempo. Es el caso de un sistema de recursos humanos de una universidad, en donde existe un plan de incentivos para su personal. Así, para aspirar a subir en este escalafón, el empleado debe cumplir con una serie de pasos, por ejemplo, la publicación de artículos, diplomas, participación en congresos, etc. El empleado interesado presenta a un Comité de Evaluación, un expediente en donde haga constar sus diferentes actividades. El Comité estudia el caso y tiempo después emite una resolución. La decisión se comunica al interesado y al Departamento de Contabilidad, para que proceda al reajuste salarial que corresponda.

Ante esta gestión, se pueden establecer varias fechas:

- Fecha en que se presentó la solicitud,
- Fecha en que se pronunció el Comité,
- Fecha en que se le avisó al interesado,
- Fecha en que se produce el reajuste, etc.

Ante una situación como la anterior, es importante contar con mecanismos que permitan desarrollar una base con datos que represente un instante del proceso o, en su defecto, contar con diferentes versiones de una misma estructura para obtener una representación más fiel de un mundo real que no es estático.

En [SNOD86] se presenta una clasificación de cómo se puede introducir el concepto de tiempo en un ambiente de bases de datos relacional, la cual se resume a continuación.

margen de los sistemas al tiempo. Sin embargo, saber cómo evoluciona el mundo puede ser una representación que representa un instante en el tiempo. Los datos pueden ser una forma de recursos humanos y de incentivos para su empleo. Al llamar a la oficina del empleado debe saber, la publicación de artículos, etc. El empleado se pone en contacto, un expediente en la oficina. El Comité estudia el expediente y la decisión se comunica a la dirección, para que proceda al despacho.

Las fechas:

Es importante contar con bases de datos que representen diferentes versiones de la misma representación más fiel de un

explicación de cómo se puede manejar el tiempo en bases de datos

En primer lugar se tienen las bases de datos relacionales tradicionales, que responden a los lineamientos que se han definido a través de los diferentes capítulos del presente libro. Una base de datos de este tipo representa una fotografía del mundo real y puede, en determinado momento, no reflejar un nuevo estado del mundo real, por haber cambiado o evolucionado la realidad. La solución que se tiene es entonces modificar la base de datos para que refleje estos posibles estados. Sin embargo, una vez que se pasa a un nuevo estado, el anterior queda "olvidado". De esta forma no existe un manejo adecuado del tiempo.

En segundo lugar, se habla de las *bases de datos de retorno*, las cuales almacenarían los estados pasados de la base de datos, indizados por el tiempo. Esto requiere una representación del tiempo transaccional, es decir, del tiempo en que los datos fueron almacenados o modificados en la base de datos. De esta forma, es posible tener acceso a un estado anterior de la base de datos moviéndose a través del eje del tiempo. Para ello, se debe implementar una operación de selección de los diferentes estados almacenados. Dicha operación puede denominarse de *retorno* -en inglés rollback-.

Ejemplo. La consulta:

Dar el salario de enero de 1990 que tuvo el empleado Carlos

utilizando una operación de retorno en un lenguaje de consultas del tipo SQL, se podría expresar de la siguiente forma:

```
select      salario
from       empleado
where      nombre = 'Carlos'
and        e2.nombre = 'Juan'
as of      'enero de 1990';
```

Un problema con este tipo de base de datos es que no se puede modificar el pasado si se descubre un error posterior. Además, un sistema tal requeriría grandes capacidades de procesamiento y de

almacenamiento para guardar los diferentes estados de la base de datos en un determinado período de tiempo.

Otra opción al manejo del tiempo se refiere a las bases de datos históricas. Una *base de datos histórica* almacena un estado por cada relación. En estos casos sí es posible corregir los errores pero no se guardan estados previos de una relación. Permiten el manejo del *tiempo válido*, es decir, el tiempo en que la relación se toma como válida. En el caso de la relación EMPLEADO del ejemplo anterior, el tiempo válido es el momento en que empieza a regir un aumento en el salario, en contraposición al tiempo transaccional que es la fecha en que se almacenaron los datos respectivos.

Ejemplo. La consulta:

Dar el salario de Carlos cuando Juan fue contratado por la empresa.

por medio de un lenguaje de tipo SQL podría atenderse de la siguiente forma:

```
select    e1. salario, e2 salario
from      empleado e1, empleado e2
where     e1.nombre = 'Carlos'
and       e2.nombre = 'Juan'
when     e1 overlap begin e2;
```

En este caso la cláusula when especifica la relación temporal de las tuplas que participan en la derivación.

Finalmente, se tienen las *bases de datos temporales* que soportan no solo el tiempo transaccional sino también el tiempo válido en una misma relación. Esto significa que para cada relación se tendrá una sucesión de históricos. Con una base de datos temporal, las tuplas pueden verse como válidas en cualquier momento, además captura los cambios retroactivos o proactivos que pueden darse.

Ejemplo. Considerar la siguiente consulta

Dar el salario Carlos cuando Juan fue contratado, según el estado de la base de datos en enero de 1990

dos de la base de datos

e a las bases de datos
una un estado por cada
los errores pero no se
en el manejo del *tiempo*
oma como válida. En el
terior, el tiempo válido
ento en el salario, en
s la fecha en que se

por la empresa.

de la siguiente forma:

elación temporal de las

orales que soportan no
pó válido en una misma
tendrá una sucesión de
as tuplas pueden verse
s captura los cambios

gún el estado de la base de

La respuesta a la misma se podría presentar de la siguiente forma:

```
select    e1. salario, e2 salario
from      empleado e1, empleado e2
where     e1.nombre = 'María'
and       e2.nombre = 'Juan'
when     e1 overlap begin e2
as of      'enero de 1990';
```

En este caso, lo que se solicita es el salario de Carlos cuando inició en la empresa, de acuerdo con el estado de la base de datos en marzo de 1990.

La mayoría de la investigación que se ha hecho en el campo temporal concierne al modelo relacional. El esfuerzo se ha centrado en el desarrollo de lenguajes de consultas temporales, así como la ampliación del modelo con el fin de incorporar el concepto de tiempo. Esto conlleva al cuestionamiento de una serie de hechos consolidados en el mundo relacional clásico.

En primer lugar, es el referente al almacenamiento de múltiples versiones de una misma estructura. Esto implica dispositivos de almacenamiento no tradicionales como los discos ópticos. En segundo lugar, los métodos de acceso deben ser adaptados para no crear *overhead*, por medio de índices secundarios o encadenamientos hacia atrás.

BIBLIOGRAFIA

- [O1IN92] 01 Informatique, Semanario, N° 1218, 26 de junio de 1992.
- [ABDE91] M. Abdelguerfi, A.K. Sood, Database Machines: Trends and Opportunities, *IEEE Micro*, vol. 11, no. 6, december, 1991.
- [ABIT87] S. Abiteboul , R. Hull, IFO: A Formal Database Model, *ACM Transactions on Database Systems*, vol. 12, no. 4, 1987.
- [ABIT89] S. Abiteboul, R. Hull, Restructuring hierarchical database objects, *Theoretical Computer Science*, vol. 62, pp. 3-38, 1988.
- [ACSI90] ACSIOME, *Modélisation dans la conception des systèmes d'information*, Masson, Paris, 1990.
- [ADIB78] M. Adiba, *Un modèle relationnel et une architecture pour les systèmes de bases de données réparties. Applications au projet POLYPHEME*. Thèse de doctorat d'état, Université Scientifique et Médicale de Grenoble, septembre, 1978.
- [AGRA94] D. Agrawal, A. El Abbadi, A.E. Lang, The Performance of Protocols Based on Locks with Ordered Sharing, *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 5, october, 1994.
- [AHME91] R. Ahmed et al, The pegasus heterogeneous Multidatabase System, *IEEE Computer*, pp. 19-27, december 1991.
- [AKOK84] J. Akoka, *Les Systèmes de Gestion de Bases de Données: Théorie et Pratique*, Eyrolles, Paris, 1984.

- [ALAG86] S. Alagic, *Relational Database Technology*, Springer-Verlag (TMCS), New York, 1986.
- [ANDL92] P. K. Andleigh, M.R. Gretzinger, *Distributed Object-Oriented Data-Systems Design*, Prentice-Hall, NJ, 1992.
- [ANSI92] American National Standards Institute (ANSI), Documents ANSI X3.135-1992.
- [ARIA86] G. Ariav, A Temporally Oriented Data Model, *ACM Transactions on Database Systems*, vol. 11, no. 4, december 1977.
- [ARIA91] G. Ariav, Temporally oriented data definitions: managing schema evolution in temporally oriented databases *Data & Knowledge Engineering*, vol. 6, pp. 451-467, 1991.
- [ARMS74] W.W. Armstrong, Dependency structures of data base relationships. *Proceedings, IFIP Congress 1974*, North Holland, 1974.
- [ASTR76] M.M. Astrahan et al., System: R Relational approach to database management, *ACM Transactions on Database Systems*, vol. 1, no 2, 1976.
- [ATKI90] M. Atkinson et al, The Object-Oriented Database System Manifesto. *Deductive and Object-Oriented Databases*, Elsevier Science Publishers, Amsterdam, Netherlands, 1990.
- [ATTA95] G.K. Attaluri et al., The CORDS multidatabase project, *IBM Systems Journal*, vol. 34, no. 1, 1995.
- [ATZE93] P. Atzeni, V. De Antonellis, *Relational Database Theory*. Benjamin/Cummings, California, 1993.
- [BANC85] F. Bancilhon, W. Kim, H. Korth, A model of CAD transactions. *Proceedings of the 11th International Conference on Very Large Databases*, Morgan Kaufmann Eds., august 1985.
- [BANC90a] F. Bancilhon, W. Kim, Object-oriented database systems: In transition, *ACM Sigmod Record*, vol. 19, no. 4, december 1990.
- [BANC90b] F. Bancilhon, S. Gamerman, Les systèmes de gestion de bases de données orientées objets, *Génie Logiciel & Systèmes Experts*, no. 20, septembre 1990.
- [BANC92] F. Bancilhon, Understanding Object-Oriented Database Systems. *Proceedings in the 3rd International Conference on Extending Database Technology*, Vienna, Austria, march 1992, LNCS 580.

- [BARB92] D. Barbará, H. García-Molina, The Demarcation Protocol: A Technique for Maintaining Linear Arithmetic Constraints in Distributed Database Systems, *Proceedings in the 3rd International Conference on Extending Database Technology*, Vienna, Austria, march 1992, LNCS 580.
- [BARG91] N.S. Barghouti, G.E. Kaiser, Concurrency Control Advanced Database Applications, *ACM Computing Surveys*, vol. 23, no. 3, september 1992.
- [BARR87] O. Barros, Information requirements and alternatives in information system design, *Information Systems*, vol. 12, no. 2, 1987.
- [BATI84] C. Batini, M. Lenzerini, A methodology for data schema integration in the entity relationship model, *IEEE Transactions on Software Engineering*, vol. SE-10, no. 6, 1984.
- [BATI92] C. Batini, S. Ceri, S. Navathe, *Conceptual Database Design*, Benjamin/Cummings, Redwood City, California, 1992.
- [BELL 92] D. Bell, J. Grimson, *Distributed Database Systems*, Reading, Addison-Wesley, Massachusetts, 1992.
- [BERN81] P.A. Bernstein, N. Goodman, Concurrency Control in Distributed, *Computing Surveys*, vol. 13, no. 2, 1981.
- [BERN87] P.A. Bernstein et al, *Concurrency Control and Recovery in Databases Systems*, Addison-Wesley, Reading, Massachusetts, 1987.
- [BERR77] C. Berri, R. Fagin, J.H. Howard, A complete axiomatization for functional and multivalued dependencies in database relations, *Proceedings ACM-SIGMOD*, Toronto, Canada, 1977.
- [BERT91a] E. Bertino, P. Foscoli, Index Organizations for Object-Oriented Systems, *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 2, april, 1995.
- [BERT91b] E. Bertino, L. Martino, Object-oriented database management systems, *IEEE Computer*, vol. 24, no. 4, april 1991.
- [BERT95] E. Bertino, G. Guerrini, D. Montesi, Towards Deductive Object Databases, *Theory and Practice of Object Systems*, vol. 1, no. 1, 1995.
- [BLAH88] M.R. Blaha, W.J. Premerlani, J.E. Rumbaugh, Relational Database Design Using an Object-Oriented Methodology, *Communications of the ACM*, vol. 31, no. 4, april, 1988.

- [BLAH94] M.R. Blaha, W.J. Premerlani, H. Shen, Converting OO Models into RDBMS Schema, *IEEE Software*, vol. 11, no. 3, may, 1994.
- [BLAS77] M.W. Blasgen, K.P. Eswaran, Storage and access in relational data bases, *IBM Systems Journal*, vol.16, no. 4, 1977.
- [BLAS81] M.W. Blasgen et al, System R: An architectural overview, *IBM Systems Journal*, vol. 20, no. 1, 1981.
- [BONT96] C. Bontempo, C. Saracco, Join Processing: The Relational Embrace. *Database Programming & Design*, vol. 9, no. 1, january 1996.
- [BOOC86] G. Booch, Object Oriented Development, *IEEE Transactions on Software Engineering*, vol. SE-12, no. 2, february, 1986.
- [BOOC91] G. Booch, *Object Oriented Design with Applications*. Benjamin/Cummings, California, 1991.
- [BOOC95] G. Booch, J. Rumbaugh, *Unified Method*, Relational Software Corporation, 1995.
- [BOSC95] P. Bosc, O. Pivert, SQLF: A relational database language for fuzzy querying, *IEEE Transactions on Fuzzy Systems*, vol. 3., no. 1, february, 1995.
- [BOUZ91] M. Bouzeghoub, G. Gardarin, E. Métais, Database design tools: An expert system approach, *Proceedings of VLDB 85*, Stockholm, 1985.
- [BROD84] M.L. Brodie, D. Ridjanovic, *Fundamentals concepts for modeling of objects*, Computer Corporation of America, Four Cambridge Center, october 1984.
- [BREI86] Y. Breitbrat, P.L. Olson, G.R. Thomson, Database Integration in a Distributed Heterogeneous Database System, *IEEE Conference on Data Engineering*, february, 1986.
- [BRUN91] J. Brunet, *O*: a model for object-oriented analysis*, rapport interne Laboratoire MASI, Université Paris VI, 1991.
- [BUCK82a] B. P. Buckles, F. E. Petry, Fuzzy databases and their applications. M.M. Gupta and E. Sánchez (Eds), *Fuzzy Information and Decision Processes*, North-Holland, Amsterdam, 1982.
- [BUCK82b] B. P. Buckles, F. E. Petry, A fuzzy representation of data for relational databases, *Fuzzy Sets and Systems*, vol. 7, no. 3, pp. 213-226, 1982.
- [BUTT91] P. Butterworth, A. Otis, J. Stein, The Gemstone Object Database Management System, *Communications of the ACM*, vol. 34, no. 10, pp. 64-77, october, 1991.

- [CANA74] R.E. Canaday et al., A back-end computer for database management, *Communications of the ACM*, vol. 17, no. 10, october 1974.
- [CARE90] M. Carey, L. Haas, Extensible Database Management Systems, *SIGMOD RECORD*, vol. 19, no. 4, december 1990.
- [CATE91] R.G.G. Cattell, *Object Data Management: Object-Oriented and Extended Relational Database Systems*, Addison-Wesley, Massachusetts, 1991.
- [CAVA93] A. Cavarero, C. González, A design process of databases on a fuzzy object-oriented approach, *Fourth Australian Database Conference*, Brisbane Queensland, Australia, 1-2 February, 1993.
- [CERI84] S. Ceri, G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, New York, 1984.
- [CHAM76] D.D. Chamberlain et al, SEQUEL2: A Unified Approach to Data Definition, Manipulation, and Control, *IBM Journal of Research and Development*, vol. 20, no. 6, november, 1976.
- [CHAM81] Chamberlain D.D. et al, A History and Evaluation of System R, *Communications of the ACM*, vol. 24, no. 10, october, 1981.
- [CHAN73] C.L. Chang, R.C.T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.
- [CHEN76] P. Chen, The entity-relationship model-Toward a unified of data, *ACM Transactions on Database Systems*, vol. 1, no. 1, march 1976.
- [CHEN76] W. Chen, Declarative Updates of Relational Databases, *ACM Transactions on Database Systems*, vol. 20, no. 1, march 1995.
- [CHIA94] R.H.L. Chiang, T.M. Barron, V.C. Storey, Reverse engineering of relational databases: Extraction of an EER model from a relational database, *Data & Knowledge Engineering*, vol. 12, pp. 107-142, 1994.
- [COAD91] P. Coad, E. Yourdon, *Object-oriented analysis* (2nd edition), Prentice Hall, Englewood, 1991.
- [CODD69] E.F. Codd, *Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks*, IBM Research Report RJ599, august, 1969.
- [CODD70] E.F. Codd, A Relational Model of Data for Large Shared Data Banks, *Communication of the ACM*, vol. 13, no. 6, june, 1970.

- [CODD74] E.F. Codd, Seven steps to rendezvous with the casual user, *Database Management*, J.W. Klimble, K.L. Koffeman (eds.), North-Holland Publishing Company, 1974.
- [CODD75] E.F. Codd, Understanding relations, *ACM-SIGMOD*, vol. 7, no. 3-4, 1975.
- [CODD79] E.F. Codd, Extending the database relational model to capture more meaning, *ACM Transactions on Database Systems*, vol. 4, no. 4, december 1979.
- [CODD82] E.F. Codd, Relational Database: A Practical Foundation for Productivity, *Communications of the ACM*, vol. 25, no. 2, february, 1982.
- [CODD85] E.F. Codd, An evaluation scheme for DBMS that are claimed to be relational, *Computer World*, october, 1985.
- [CODD90] E.F. Codd, *The Relational Model for Database Management (version 2)*, Addison-Wesley, Reading, Massachusetts, 1990.
- [CODD93] E.F. Codd, C.J. Date, Much Ado About Nothing, *Database Programming & Design*, vol. 6, no. 10, october, 1993.
- [COHE95] H. Cohen, Los números primos, *Mundo Científico*, vol. 15, no. 161, octubre 1995.
- [COLE94] D. Coleman et al., *Object-oriented development: The Fusion method*, Prentice-Hall, NJ, 1994.
- [COPE84] G. Copeland, D. Maier, Making Smalltalk a database system, *ACM SIGMOD Record*, june 1984.
- [CORD95] L.E. Cordero, S.F Rodríguez, *Desarrollo de un lenguaje difuso de consultas: Aplicación a la inferencia de poblaciones de nemátodos en plantaciones de banano en Costa Rica*, Tesis de Maestría en Computación, Instituto Tecnológico de Costa Rica, agosto, 1995.
- [DALE90] R. Dale, Client-Server Database: Architecture of the Future, *Database Programming & Design*, vol. 3, no 8, august, 1990.
- [DARW95] H. Darwen, C.J. Date, Introducing The Third Manifesto, *Database Programming & Design*, vol. 8, no. 1, january, 1995.
- [DATE86] C.J. Date, *An Introduction to Database Systems*, Volume I (Fourth Edition), Addison-Wesley, Reading, Massachusetts, 1986.
- [DATE89] C.J. Date, *Relational Database: Select Writings*, Addison-Wesley, Reading, Massachusetts, 1989.

- [DATE94] C.J. Date, H. Darwen, D. McGoveran, Nothing to Do with the Case, *Database Programming & Design*, vol. 7, no. 7, july, 1994.
- [DATE95] C.J. Date, D. McGoveran, A New Database Design Principle, *Database Programming & Design*, vol. 7, no. 7, july, 1994.
- [DAVY94] M.M. Davydov, From Model to Database, *Database Programming & Design*, vol. 7, no. 3, 1994.
- [DAYA84] V. Dayal, H. Y. Hwang, View definition and generalization for database integration in a multibase system, *IEEE Transactions on Software Engineering.*, vol. SE-10, no. 6, october, 1984.
- [DELO78] C. Delobel, Normalization and hierarchical dependencies in the relational data model, *ACM Transactions on Database Systems*, vol. 3, no. 3, september, 1978.
- [DELO82] C. Delobel, M. Adiba, *Bases de Données et Systèmes Relationnels*, Bordas, Paris, 1982.
- [DELO91] C. Delobel et al, *Bases de Données: des Systèmes Relationnels aux Systèmes à Objets*, InterEditions, Paris, 1991.
- [DEMA78] T. de Marco, *Structured Analysis and Systems Specification*, Prentice-Hall, New Jersey, 1978.
- [DENN79] D. Denning, P. Denning, Data Security, *ACM Computing Surveys*, vol. 11, no. 3, 1979.
- [DEUX91] O. Deux et al, The O2 System, *Communications of the ACM*, vol. 34, no. 10, october 1991.
- [DIAZ94] O. Díaz, N.W. Paton, Extending ODBMSs Using Metaclasses, *IEEE Software*, vol. 11, no. 3, may, 1994.
- [DIBA93] G. Di Battista, M. Lenzerini, Deductive Entity Relationship Modeling, *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 3, 1993.
- [DIVI91] M. Diviné, *Parlez-Vous Mérise*, Eyrolles, Paris, 1991.
- [DUTK89] A.F. Dutka, H.H. Hanson, *Fundamentals of Data Normalization*, Addison-Wesley, Reading, Massachusetts, 1989.
- [EICK85] C. Eick, T. Raupp, Towards a formal semantics and inference rules for conceptual data models, *Data & Knowledge Engineering*, vol. 6, pp. 297-317, 1985.

- [ELM85] R. Elmasri R., J. Weeldreyer, A. Hevner, The Category Concept: An Extension to the Entity-Relationship Model, *Data & Knowledge Engineering*, vol. 1, pp. 75-116, 1985.
- [ELM94] R. Elmasri, S.B. Navathe, *Fundamentals of Database Systems*, Benjamin/Cummings, Redwood City, 1994.
- [EL-RE95] H. El-Rewini et al., Object Technology: A Virtual Roundtable, *IEEE Computer*, vol. 28, no. 10, october 1995.
- [EMBL92] D.W. Embley, B.D. Kurtz, S.N. Woodfield, *Object-Oriented System Analysis: A Model-Driven Approach*, Prentice-Hall, 1992.
- [EMBL95] D.W. Embley, R.B. Jackson, S.N. Woodfield, OO Systems Analysis: Is It or Isn't It ?, *IEEE Computer*, vol. 28, no. 7, july, 1995.
- [ERWI94] M. Erwig, R.H. Güting, Explicit Graphs in a Functional Model for Spatial Databases, *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 5, october, 1994.
- [FAGI77] R. Fagin, Multivalued dependencies and a new normal form for relational database design, *ACM Transactions on Database Systems*, vol. 2-3, september 1977.
- [FAUD91] P. Faudemay, M. Mhiri, An Associative Accelerator for large Databases, *IEEE Micro*, vol. 11, no. 6, december, 1991.
- [FERN81] E.B. Fernández, R.C. Summers, C. Wood, *Database Security and Integrity*, Addison-Wesley, Reading, Massachusetts, 1981
- [FICH92] R.G. Fichman, C.F. Kemerer, Object-oriented and conventional analysis and design methodologies, comparison and critique, *IEEE Computer*, vol. 25, no. 10, october 1992.
- [FLEM89] C. Fleming, B. von Halle, *Handbook of Relational Database Design*, Addison-Wesley, Reading, Massachusetts, 1989.
- [GAL 95] A. Gal, O. Etzion, Maintaining data-driven rules in databases, *IEEE Computer*, vol. 28, no. 1, january 1995.
- [GAND92] M. Gandhi, E.L. Robertson, A Specification-Based Data Model. *11th International Conference on the Entity-Relationship Approach*, Karsruhe, Germany, october, 1992 (LNCS 645).
- [GARC87] H. García-Molina, K. Salem, Sagas, *ACM SIGMOD Record*, vol. 16, no. 3, december, 1987.
- [GARC90] H. García-Molina, B. Lindsay, Research Directions for Distributed Databases, *ACM SIGMOD Record*, vol. 19, no. 4, december, 1990.

- [GARD85a] G. Gardarin, *Bases de Données : Les systèmes et leurs langages*, Eyrolles, Paris, 1985.
- [GARD85b] G. Gardarin, P. Valduriez, *Bases de Données Relationnelles: Analyse et comparaison des systèmes*, Eyrolles, Paris, 1985.
- [GARD89] G. Gardarin, P. Valduriez, *Relational Databases and Knowledge Bases*, Addison-Wesley, Reading, Massachusetts, 1989.
- [GARD90] G. Gardarin, J. Kiernan, Deductive database rule languages: Analysis and case study, *Database Systems of the 90s: International Symposium*, Berlin FRG, november, 1990, LNCS 466.
- [GEOR92] R. George, F. E. Petry, B.P. Buckles, Uncertainty modeling in object-oriented geographical information systems, submitted to *the Conference on Database and Expert Applications*, 1992.
- [GOBL92] C. Goble et al., The Manchester Multimedia Information System, *Proceedings in the 3rd International Conference on Extending Database Technology*, Vienna, Austria, March 1992, LNCS 580.
- [GOLD90] B. Gold-Bernstein, Does Client-Server Equal Distributed Database?, *Database Programming & Design*, vol. 3, no 9, september, 1990.
- [GONZ93] C. González, A. Cavarero, *Metodología de Análisis y Diseño de Bases de Datos en un Ambiente Difuso Orientado a Objetos*, Reporte Técnico CIC-RT-93-10, Instituto Tecnológico de Costa Rica, marzo, 1993.
- [GONZ95] C. González, Perspectivas en Ciencia y Tecnología de la Lógica Difusa, *Tecnología en Marcha*, vol. 12., Número Especial, 1995
- [GRAE93] G. Graefe, Query Evaluation Techniques for Large Databases, *ACM Computing Surveys*, vol. 25, no. 2, june, 1993.
- [GRAH91] I. Graham, *Object-oriented methods*, Addison Wesley, NY, 1991
- [GRAY75] N. Gray et al., *Granularity of Locks and Degree of Consistency in a Shared Data Base*, IBM-RJ1653, september, 1975.
- [GRAY78] N. Gray, Notes on database operating systems, in *Operating Systems: An Advanced Course, Lecture Notes in Computer Science 60*, Springer-Verlag, Berlin, 1978.
- [GRAY81] N. Gray et al, The Recovery manager of the System R Database Manager, *ACM Computing Surveys*, vol. 13, no. 2, june, 1981.
- [GUEN90] O. Guenther, A. Buchmann, Research Issues in Spatial Databases, *ACM SIGMOD Record*, vol. 19, no. 4, december, 1990.

- [GUPT94] Gupta, *SQLBase, SQL Reference*, Gupta Corporation, 1994.
- [HAN 95] J. Han, Chain-Split Evaluation in Deductive Databases, *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 2, april, 1995.
- [HAER83] T. Haerder, A. Reuter, Principles of Transaction-Oriented Database Recovery, *ACM Computing Surveys*, vol. 15, no. 4, december, 1983.
- [HASS89] L.M. Hass et al., Extensible Query Processing in Starburst, *Proceedings in ACM SIGMOD Record*, Portland, Oregon, 1989.
- [HAAS90] L.M. Hass et al., Starburst Mid-Flight: As the Dust Clears, *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 1, march, 1990.
- [HERZ92] R. Herzig, M. Gogolla, Transforming conceptual data models into an object model, *11th International Conference on the Entity-Relationship Approach*, Karlsruhe, Germany, october, 1992, LNCS 645.
- [HOFF70] L.J. Hoffman, W.F. Miller, Getting a personal dossier from a statistical data bank, *Datamation*, vol. 16, no. 5, may, 1970.
- [HONI93] S. Honiden, N. Kotaka, Y. Kishimoto, Formalizing Specification Modeling in OOA, *IEEE Software*, january 1993.
- [HORO92] J.R. Horowitz, A.F. Cárdenas, Decomposing Heterogeneous Inter-Entity Relationship Updates, *IEEE Transactions on Knowledge and Data Engineering*, vol. 4, no.4, august, 1994.
- [HSIA83] D. K. Hsiao, Editor, *Advanced Database Machine Architecture*, Prentice-Hall, NJ, 1983.
- [HSIA91] D. K. Hsiao, A Parallel, Scalable, Microprocessor-Based Database Computer for Performance Gains and Capacity Growth, *IEEE Micro*, vol. 11, no. 6, december, 1991.
- [HSU 88] C. Hsu et al., TSER: A Data Modeling Systems Using the Two-Stage Entity-Relationship Approach, *Entity Relationship Approach*, S.T. March (Editor), Elsevier Science Publishers B.V. (North Holland), 1988.
- [HUEM95] C. Huemer, G. Kappel, S. Vieweg, Migration in Object-Oriented Database Systems-A Practical Approach, *Software-Practice and Experience*, vol. 25, no. 10, october 1995.

- Corporation, 1994.
- ductive Databases, *IEEE Engineering*, vol. 7, no. 2, april,
- nsaction-Oriented Database
15, no. 4, december, 1983.
- Processing in Starburst,
Portland, Oregon, 1989.
- As the Dust Clears, *IEEE Engineering*, vol. 2, no. 1,
- nceptual data models into an
conference on the Entity- many, october, 1992, LNCS
- a personal dossier from a
6, no. 5, may, 1970.
- . Formalizing Specification
ary 1993.
- posing Heterogeneous Inter-
Transactions on Knowledge and
1994.
- base Machine Architecture,
- coprocessor-Based Database
Capacity Growth, *IEEE Micro*,
- Systems Using the Two-Stage
Relationship Approach, S.T.
lishers B.V. (North Holland).
- Migration in Object-Oriented
roach, *Software-Practice and*
95.
- [INFO87] INFORMIX-SQL, Reference manual, july 1987.
- [INOU91] U. Inoue et al., Rinda: A Relational Database Processor with hardware Specialized for Searching and Sorting, *IEEE Micro*, vol. 11, no. 6, december, 1991.
- [JACO85] B.E. Jacobs, *Applied Database Logic I: Fundamental Database Issues*, Prentice-Hall, New Jersey, 1985.
- [JACO92] I. Jacobson et al., *Object-Oriented Software Engineering: a use case driven approach*, Addison-Wesley, Wokingham, England, 1992
- [JAGA89] H.V. Jagadish, Incorporating hierarchy in a relational model of data, *ACM SIGMOD Record*, vol. 18, no. 2, june 1989.
- [JAGA95] V. Jagadish, The Incinerate Data Model, *ACM Transactions on Database Systems*, vol. 20, no. 1, march, 1995.
- [JARK84] M. Jarque, J. Koch, Query Optimization Systems, *ACM Computing Surveys*, vol. 16, no. 2, june, 1984.
- [JOHN95] T. Johnston, More to the Point, *Database Programming & Design*, vol. 8, no. 11, november, 1995.
- [JOSE91] J. V. Joseph, S. M. Thatte, C.W. Thompson, D. C. Wells, Object-oriented databases : design and implementation, *Proceedings of THE IEEE*, vol. 79, no. 1, 1991.
- [KACP86] J. Kacprzyk, A. Ziolkowski, Database queries with fuzzy linguistic quantifiers, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 3, 1986.
- [KELL86] A.M. Keller, The Role of Semantics in Translating View Updates, *IEEE Computer*, vol. 19, no.1, january, 1986.
- [KEMP87] A. Kemper, M. Wallrath, An Analysis of Geometric Modeling in Database Systems, *ACM Computing Surveys*, vol. 19, no. 1, march 1987.
- [KENT83] W. Kent, A simple guide to five normal forms in relational database theory, *Communications of the ACM*, vol. 26, no. 2, february, 1983.
- [KIM 79] W. Kim, Relational Database Systems, *ACM Computing Surveys*, vol. 11, no. 3, september, 1979.
- [KIM 90a] W. Kim, *Introduction to Object-Oriented Databases*, MIT Press, Cambridge, 1990.

- [KIM90b] W. Kim et al., Architecture of the Orion Next-Generation Database System, *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 1, march, 1990.
- [KIM 91] W. Kim, Classifying schematic and data heterogeneity in multidatabase systems, *IEEE Computer*, vol. 24, no. 12, 1991.
- [KLIR95] G.J. Klir, B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice-Hall, NJ, 1995.
- [KORN94] Y. Kornatzky, S.E. Shimony, A Probabilistic Object-Oriented Data Model, *Data & Knowledge Engineering*, vol. 12, pp. 143-166, 1994.
- [KORT86] H.F. Korth, A. Silberschatz, *Database System Concepts*, McGraw-Hill, New York, 1986.
- [KROH93] P. Kroha, *Objects and databases*, McGraw-Hill, New York, 1993.
- [KU 94] C.S. Ku, H.D. Kim, L.J. Henshen, An Efficient Indefiniteness Inference Scheme in Indefinite Deductive Databases, *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 5, october, 1994.
- [KUPE93] G.M. Kuper, M.Y. Vardi, The Logical Data Model, *ACM Transactions on Database Systems*, vol. 18, no. 3, september 1993.
- [LAMB91] C. Lamb et al., The Objectstore Database System, *Communications of the ACM*, vol. 34, no. 10, october, 1991.
- [LARS89] J. A. Larson, S. B. Navathe, A theory of attribute equivalence in databases with application to schema integration, *IEEE Transactions on Software Engineering*, vol. SE-15, no. 4, april, 1989.
- [LOCK90] C. Lockemann, A. Kemper, G. Moerkotte, Future Database Technology: Driving Forces and Directions, *Database Systems of the 90s: International Symposium*, Berlin FRG, november, 1990, LNCS 466.
- [LOHM91] G.M. Lohman et al., Extension to Starburst: Objects, Types, Functions and Rules, *Communications of the ACM*, vol. 34, no. 10, october, 1991.
- [LOOM87] M.E. Loomis, A. V. Shaw, J. Rumbaugh, An object modeling technique for conceptual design, *European Conference on Object Oriented Programming*, Paris, France, published as *Lecture Notes in Computer Science*, 276, Springer-Verlag, pp. 192-202, 1987.

- Next-Generation Database
and Data Engineering, vol.
- 1 data heterogeneity in
vol. 24, no. 12, 1991.
- Fuzzy Logic: Theory and
- listic Object-Oriented Data
vol. 12, pp. 143-166, 1994.
- System Concepts, McGraw-
- w-Hill, New York, 1993.
- An Efficient Indefiniteness
ductive Databases, *IEEE*
Engineering, vol. 6, no. 5,
- ta Model, *ACM Transactions*
tember 1993.
- use System, *Communications*
01.
- y of attribute equivalence in
ntegration, *IEEE Transactions*
o. 4, april, 1989.
- oerckotte, Future Database
ons. *Database Systems of the*
FRG, november, 1990, LNCS
- Starburst: Objects, Types,
of the ACM, vol. 34, no. 10.
- baugh, An object modeling
opean Conference on Object
published as *Lecture Notes in*
g, pp. 192-202, 1987.
- [LOOM95] M.E. Loomis, *Object databases: the essentials*, Addison Wesley, MA, 1995.
- [LUNT90] T.F Lunt, E.B. Fernández, Database Security, *ACM SIGMOD Record*, vol. 19, no. 4, december 1990.
- [LYYT87] K. Lyytinen, Different perspectives on Information Systems: Problems and Solutions, *ACM Computing Surveys*, vol. 19, no. 1, march, 1987.
- [MAIE83] D. Maier, *The Theory of Relational Databases*, Computer Science Press, Maryland, 1983.
- [MAKI77] A. Makinouchi, A consideration on normal form of not-necessary-normalized in the relational model of data, *Proc. Intl Conf. on Very Large Data Bases*, 1977.
- [MALY90] T. A. Malyuta, V. V. Pasichnik, A.A. Stogniy, Means for management of relational fuzzy data bases-way to merging of systems of data bases and knowledge bases, Database Systems of the 90's International Symposium, Berlin, November 1990, published as *Lecture Notes in Computer Science*, 460, pp. 337-346, 1991.
- [McGE77] W.C. McGee, The information management system IMS/VS. Part I: General structure and operation, *IBM Systems Journal*, vol. 16, no. 2, 1977.
- [MEYE92] K. Meyer-Wegener, Multimedia databases: integrated storage and retrieval of text, images, sound and video, *Institut für Mathematische Maschinen und Datenverarbeitung, Informatik, Friedrich Alexander Universität*, vol. 25, no. 12, november, 1992.
- [MIRA84] S. Miranda, J. M. Busta, *L'Art des Bases de Données*, vol. 1 *Introduction aux Bases de Données*, Eyrolles, 1984.
- [MIRA86] S. Miranda, J. M. Busta, *L'Art des Bases de Données*, vol. 2 *Les Bases de Données Relationnelles*, Eyrolles, 1986.
- [MIRA88] S. Miranda, *Comprendre et Concevoir les Bases de Données Relationnelles*, Editions P.S.I., Paris, 1988.
- [MISH92] P. Mishra, M.H. Eich, Join Processing in Relational Databases, *ACM Computing Surveys*, vol. 24, no. 1, march ,1992.
- [MORR90] J. M. Morressey, Imprecise information and uncertainty in information systems, *ACM Transactions on Information Systems*, vol. 8, no. 2, 1990.

- [MOTS95] R. Motschinig-Pitrik, V.C. Storey, Modeling of set membership: The notion and the issues, *Data & Knowledge Engineering*, vol. 16, no. 2, august, 1995.
- [NAVA92] S.B. Navathe, The next ten years of modeling, methodologies and tools, *11th International Conference on the Entity-Relationship Approach*, Karlsruhe, Germany, october, 1992. LNCS 645.
- [NEGO85] C. V. Negoita, *Expert systems and fuzzy systems*, Benjamin/Cummings: California, 1985.
- [NIJS89] G.M. Nijesen, T.A. Halpin, Conceptual Schema and Relational Database Design, Prentice Hall of Australia Pty Ltd, 1989
- [NOLL91] J. Noll, W. Scacchi, Integrating Diverse Information Repositories: A Distributed Hypertext approach, *IEEE Computer*, vol. 24, no. 12, december 1991.
- [OGLE 95] V.E. Ogle, M. Stonebraker Chabot: Retrieval from a relational database of images, *IEEE Computer*, vol. 28, no. 9, 1995.
- [OLSO95] J. Olson, Building a Database Topology Strategy, *Database Programming & Design*, vol. 8, no.6, 1995.
- [OOPS91] OOPSLA, Worshop: Finding the object, addendum to the Proceedings in *OOPSLA/ECOOP'90*, Ottawa, Canada, 1991.
- [OREN92] J. Orenstein et al., Query Processing in the ObjectStore Database System, *Proceedings in ACM SIGMOD Record*, 1992.
- [OZKA86] E. Ozkarahan, *Database Machines and Database Management*, Prentice-Hall, NJ, 1986.
- [PAGE92] J. Page, A Study of Parallel Database Machine and its Performance The NCR/Teradata DBC/1012, *Proceedings of the 10th British National Conference on Databases, BNCOD*, Aberdeen, Scotland, July 1992, LNCS 618.
- [PARE88] C. Parent, S. Spaccaprieta, Gestion d'objets complexes avec des entités complexes, *Journées AFCET Bases de Données Déductives et Bases de Données Orientées Objets*, 13-14 décembre, 1988.
- [PARE89] J. Paredaens et al, *The Structure of the Relational Database Model*, Springer-Verlag (EATCS #17), Berlin, 1989.
- [PARS89] K. Parsaye et al, *Intelligent Databases*, John Wiley & Sons, New York, 1989.

- [PECK88] J. Peckman, Semantic data models, *ACM Computing Surveys*, vol. 20, no. 3, september, 1988.
- [PITO95] E. Pitoura et al., Object Orientation in Multidatabase Systems, *ACM Computing Surveys*, vol. 27, no. 2, june, 1995.
- [PONC91] P. Poncelet et al., *De HOOD à HOOK: Une méthode de conception uniforme pour ADA et les Bases de Données Orientées Objets*, Rapport K-I2S/HOOK, Université de Nice Sophia Antipolis, janvier 1991.
- [PRAM92] S. Pramanik, D. Vineyard, Fragmentation of Recursive Relations in Distributed Databases, *Proceedings in the 3rd International Conference on Extending Database Technology*, Vienna, Austria, March 1992, LNCS 580.
- [RAJU88] K.V. Raju, A.K. Majumdar, Fuzzy Functional Dependencies and Lossless Join Decomposition of Fuzzy Relational Database Systems, *ACM Transactions on Database Systems*, vol. 13, no. 2, june 1988.
- [RAO 94] B. R. Rao, *Object-oriented databases: technology, applications and products*, McGraw-Hill, New York, 1994.
- [RODG90] U. Rodgers, *Unix Database Management Systems*, Yourdon Press, Englewood Cliffs, New Jersey, 1990.
- [RISH92] N. Rishe, *Database Design: The semantic modeling approach*, McGraw-Hill, New York, 1992.
- [RITH80] J.B. Rothnie et al., Introduction to a System for Distributed Databases (SDD-1), *ACM Transactions on Database Systems*, vol. 5, no.1, march, 1980.
- [ROLL90] C. Rolland, A. Flory, Conception des systèmes d'information: Etat de l'art et nouvelles perspectives, *Inforsid*, Paris, 1990
- [RUMB91] J. Rumbaugh et al, *Object-oriented modeling and design*, Prentice-Hall, New Jersey, 1991.
- [RUND92] E.A. Rundensteiner, L. Bic, Set Operations in Object-Based Data Models, *IEEE Transactions on Knowledge and Data Engineering*, vol. 4, no. 3, 1992.
- [RUSI91] M. Rusinkiewicz, a. Sheth, G. Karabatis, Specifying Interdatabase dependencies in a Multidatabase Environment *IEEE Computer*, vol. 24., no. 12, december 1991.
- [SANS88] J.P. Sansonnet, L'architecture des nouveaux ordinateurs, *La Recherche*, no. 204, 1988.

- [SCHE90] H.J. Schek, M.H. Scholl, Evolution of Data Models, *Database Systems of the 90s: International Symposium*, Berlin FRG, november, 1990, LNCS 466.
- [SCHI85] U. Schiel, The time dimension in information systems, *Information Systems: Theoretical and Formal Aspects*, Sernadas A., Bubenko J., Olivé A., (Eds.), Elsevier Science Publishers B.V. (North Holland) IFIP, 1985
- [SCHL75] J. Schlörer, Identification and retrieval of personal records from statistical data bank, *Methods Inf. Med.*, vol. 14, no. 1, january, 1975.
- [SERN91] C. Sernadas, J. Fiadero Towards Object-Oriented Conceptual Modeling, *Data & Knowledge Engineering*, vol. 6, pp. 479-508, 1991.
- [SHAL91] T. Shalom, Some Recent Developments in deductive databases, *ACM SIGMOD Record*, vol. 20, no. 4, december, 1991.
- [SHAW90] P. Shaw, Database Language Standards: Past, Present, and Future, *Database Systems of the 90s: International Symposium*, Berlin FRG, november, 1990, LNCS 466.
- [SHEN90a] S. Shenoi, A. Melton, L. T. Fan, An equivalence classes model of fuzzy relational databases, *Fuzzy Sets and Systems*, vol. 38, 1990.
- [SHEN90b] S. Shenoi, A. Melton, An extended version of the fuzzy relational database model, *Information Sciences*, vol. 52, 1990.
- [SHLA88] S. Shlaer, S. Mellor, *Object-Oriented Systems Analysis*, Prentice Hall, NJ, 1988.
- [SHOV91] P. Shoval, S. Zohn, Binary-relationship integration methodology, *Data & Knowledge Engineering*, North Holland, vol. 6, 1991.
- [SILB91] A. Silberschatz, M. Stone Braker, J. Ullman Eds., Database Systems: Achievements and Opportunities, *Communications of the ACM*, vol. 34, no. 10, october, 1991.
- [SILB96] A. Silberschatz, M. Stone Braker, J. Ullman Eds., Database Research: Achievements and Opportunities Into the 21st Century, *ACM SIGMOD Record*, vol. 25, no. 1, march, 1996.
- [SMIT75] J. Smith, P. Chang, Optimizing the Performance of a Relational Algebra Interface, *Communications of the ACM*, vol. 18, no. 10, october, 1985.

- [SMIT85] H.C. Smith. Database design: Composing fully normalized tables from a rigorous dependency diagram. *Communications of the ACM*, vol. 28, no. 8, august, 1985.
- [SNOD90] R. Snodgrass. Temporal Databases, *IEEE Computer*, vol. 16, no. 9, september, 1986.
- [SNOD90] R. Snodgrass, Temporal Databases: Status and Research Directions, *ACM SIGMOD Record*, vol. 19, no. 4, december, 1990.
- [SONG92] W.W. Song, P. Johannesson, J.A. Bubenko, Semantic Similarity Relations in Schema Integration, *11th International Conference on the Entity-Relationship Approach*, Karsruhe, Germany, october, 1992 LNCS 645.
- [SOPA91] N. Soparkar, H.F. Korth, A. Silberschatz, Failure-Resilient Transaction management in Multidatabases, *IEEE Computer*, vol. 21, no. 12, december 1991.
- [SPAC94] S. Spaccapietra, C. Parent, View Integration: A Step Forward in Solving Structural Conflicts, *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no.2, august, 1994.
- [SPYR87] N. Spyros, The Partition model: a deductive database model, *ACM Transactions on Database Systems*, vol. 12, no. 1, march 1987.
- [SQLB94] *SQLBase, Reference manual*, Gupta Corporation, 1994.
- [STAN86] W. Staniszakis, Integrating heterogeneous databases, *State of the Art Report on Relational Databases*, Pergamon Press Ltd., 1986.
- [STON76] M. Stonebraker, E. Wong, P. Kreps, The Design and Implementation of INGRES, *ACM Transactions on Database Systems*, vol. 1, no. 3, september, 1976.
- [STON91] M. Stonebraker, Greg Kemnitz, The Postgres Next-Generation Database Management System, *Communications of the ACM*, vol. 34, no. 10, october, 1991.
- [STON94] M. Stonebraker (ed.), *Readings in Database Systems*, Morgan Kaufmann Publishers, California, 1991.
- [STRA95] D.D. STRAUBE. M.T. Özsü, Query Optimization and Execution Plan Generation in Object-Oriented Database Systems, *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 2, april, 1995.
- [SYBA88] Sybase Inc., SYBASE Compliance with Codd's Rules for Relational Database Systems, documentación interna, 1988.

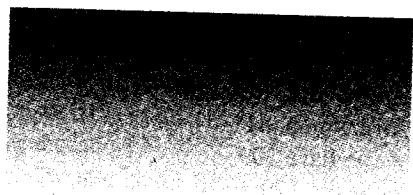
- [TARI92] Z. Tari, On the design of object-oriented databases, *11th Int. Conference on the Entity-Relationship Approach*, Karlsruhe, Germany, october, 1992, LNCS 645.
- [TEMP87] M. Templeton et al., Mermaid-A Front-End to Distributed Heterogeneous Databases, *Proceedings IEEE*, vol. 75, no. 5, may, 1987.
- [THOM89] D. Thomas, What's an object ?, *Byte*, march, 1989.
- [THOM90] G. Thomas et al., Heterogeneous Distributed Database Systems for Production Use, *ACM Computing Surveys*, vol. 22, no. 3, september, 1990.
- [THOM95] A. Thomasian, Checkpointing for Optimistic Concurrency Control Methods, *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 2, april, 1995.
- [THUR95] B. Thurasingham, W. Ford, Security Constraint Processing in a Multilevel Secure Distributed Database Management System, *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 2, april, 1995.
- [TSIC82] D.C. Tsichritzis, F.H. Lochovsky, *Data Models*, Prentice-Hall, New Jersey, 1982.
- [TSUR91] S. Tsur, Some Recent Developments in Deductive databases, *ACM SIGMOD Record*, vol. 20, no. 4, december, 1991.
- [ULMA90] J.D. Ullman, C Zaniolo, Deductive Databases: Achievements and Future Directions, *ACM SIGMOD Record*, vol. 19, no. 4, december, 1990.
- [UNLA90] R. Unland G. Schlageter, Object-Oriented Database Systems: Concepts and Perspectives, *Database Systems of the 90s: International Symposium*, Berlin FRG, november, 1990, LNCS 466.
- [[UNWA92]] M. Unwalla, J. Kerridge, Control of a Large Massively Parallel Database Machine Using SQL Catalogue Extensions, and a DSDL in Preference to an Operating System, *Proceedings of the 10th British National Conference on Databases, BNCOD*, Aberdeen, Scotland, July 1992, LNCS 618.
- [VOSS90] G. Vossen, *Data Models: Database Languages and Database Management Systems*, Addison-Wesley, Reading, Massachusetts, 1981.

- [VOSS91] G. Vossen. Bibliography on Object-oriented Database Management, *ACM SIGMOD Record*, vol 20, no. 1, march, 1991.
- [WAI 88] G. Wai. How to Choose an RDBMS: A practical guide for decision makers. Informix Software Inc., 1988.
- [WEGN95] P. Wegner, A Perspective on Object-Oriented Research, *Theory and Practice of Object Systems*, vol. 1, no. 2, 1995.
- [WILL82] R. Williams et al., R*: An Overview of the Architecture, *Proc. Int. Conf. on Database Systems*, Jerusalem, 1982.
- [WINT94] R. Winter, The Future of Very Large Database, *Database Programming & Design*, vol. 7, no. 12, 1994.
- [WIRF90] R. Wirfs-Brock, *Designing Object-Oriented Software*, Prentice-Hall, NJ, 1990.
- [YEH 87] S. Yeh, C. Ellis, A. Ege, H. Korth, *Performance analysis of two concurrency control schemas for design environments*, Technical Report, STP-036-87, MCC, Austin, Texas, 1987.
- [YU 91] C. Yu et al., Determining Relationship among Names in Heterogeneous Databases, *ACM SIGMOD Record*, vol. 20, no. 4, 1991.
- [ZADE65] L. Zadeh, Fuzzy Sets, *Information and Control*, vol. 8, 1965.
- [ZADE86] L. Zadeh, Test-score semantics as a basis for a computational approach to the representation of meaning, *Literary and Linguistic Computing*, vol. 1, 1986.
- [ZADE88] L. Zadeh, Fuzzy logic, *IEEE Computer*, april, 1988.
- [ZADE89] L. Zadeh, Knowledge representation in fuzzy logic, *IEEE Trans. on Knowledge and Data Eng.*, vol. 1, no. 1, march, 1989.
- [ZANI77] C. Zaniolo, Analysis and design for relational schemata for database systems, Doctoral dissertation, Computer Science Department, University of California, Los Angeles, UCLA-ENG-7669, 1976.
- [ZIMM87] H. J. Zimmermann, *Fuzzy set theory and its applications*, Dordrecht, The Netherlands: Nijhoff, 1987.
- [ZLOO77] M.M. Zloof M., Query-by-Example: a database language, *IBM Systems Journal*, vol.16, no. 4, 1977.

Lista parcial de revistas dedicadas a la tecnología de bases de datos o en las cuales se publican artículos de este campo con cierta frecuencia:

- ACM Computing Surveys
- ACM SIGMOD Record
- ACM Transaction on Database Systems
- ACM Transaction on Office Information Systems
- Communications of the ACM
- Computer Journal
- Data & Knowledge Engineering
- Database Programming & Design
- IEEE Computer
- IEEE Transactions on Knowledge and Data Engineering
- IEEE Transactions on Software Engineering
- Information Systems
- Journal of Computer and System Sciences
- Journal of the ACM
- Software-Practice and Experience
- Theory and Practice of Object Systems

ases de datos o en la cuales



INDICE TEMÁTICO

eering

a

ABD 40
Abstracción 405
ADN 55, 442
Administrador de
comunicación 388
datos 356
interfaces 387
memoria asociativa 388
optimizador de consulta 388
transacciones 44, 356
Agregación 71
Algebra
de herencias 241
de composiciones 242
relacional 89
Algoritmo de optimización 269
Alter table 133
add 133
drop 133
modify 133

Análisis estructurado moderno 196
Analista de datos 39
Analista estratégico 39
ANSI/SPARC 41, 127
ANSI/SQL 33
Archivo directo 21
Archivo secuencial 20
Armazón 81
Arquitectura
MIMD 392
MISD 391
SIMD 391
SISD 390
Asc 147
Asociación 60, 205
binaria 61
cardinalidad 64
muchos a uno 62
muchos a muchos 62
n-aria 61
redundante 239

- uno a uno 61
uno a muchos 61
Atomo 108, 109, 114
Atributo 65, 81
 no llave 209
Aumento 170
Avg 144
Axioma de inferencia 170
- b**
- Back-end 378
Base de datos 19 (ver SBD)
Bitácora 38, 345
Bloque
 de calificación 128, 136
 sfw 128, 136
Bloqueo generoso 308
Buffer de tabla de página 341
Buffer de página 341
- c**
- Cálculo de predicados 107
 de tuplas 107, 114
 de dominios 107, 109
Cardinalidad de asociación 64
Catálogo 128
Categoría 70
Celda 81
Cerebro 55
Cerradura 172
Cerrojo 294
 compartido 297
 exclusivo 294
 intensión 304
Ciclo de vida 195
Cinta magnética 20
Clase de objetos 403
Cláusula de base de datos 448
Cláusula de Horn 448
Clave pública 330
- RSA 330
Cliente 45
Cliente/Servidor 44
Cobertura 174
 mínima 175
Código 36, 328
Código genético 442
Columna 130
Commit 285
Componente 71, 82
Comportamiento 402
Composición 71, 204
Compuesto 71
Computación cooperativa 45
Condición de selección 97
Conditions 113
Conflicto 35
Conocimiento 55
 colectivo 55
Contenido 82
Consulta 38
 declarativa 29
Control
 acceso 36, 315
 conurrencia 34
 criptográfico 316
 multi-nivel 36, 315, 322
 inferencia 36, 316, 325
Costo de
 acceso memoria principal 251
 almacenamiento 252
 comunicación 251
 procesamiento 252
Count 144
Create
 database 130
 index 150
 schema 131
 table 131
 trigger 153

dor 44

4

75

28

tico 442

)

71, 82

ento 402

71, 204

1

cooperativa 45

selección 97

13

o 55

55

a 29

. 315

cia 34

co 316

1 36, 315, 322

36, 316, 325

memoria principal 251

uiente 252

ión 251

ento 252

30

1

3

Criptografía 327
Criptograma 36, 328
Cuneiforme (escritura) 19
Cullinet 26

d

Dato 55
datum 55
Delete from where 135
Dependencia funcional 167
 elemental 175
 no redundante 175
Dependencia multivaluada 184
Dependencia parcial 177
Dependencia producto 186
Dependencia transitiva 179
Dependiente 167
Derivación 171
Desc 147
Descomposición de consultas 270
Desnormalización controlada 221
Determinante 167
Diagrama entidad-asociación 196
Diccionario de datos 199
 dependiente 200
 independiente 200
Diferencia 91
Disco magnético 20
Disco óptico 444
Diseño 195
Disparador 332
División 104, 143
Dominio 81
 de atributo 209
 de relación 81
 del problema 57
 primario 85
Drop
 table 134
 view 150

e
Ejecución atómica 37
Emigración (atributo) 177
Emisor 317
Encabezado 82
Encapsulamiento 406
Encriptamiento 36
Entidad 59

 sub- 68
 super- 68

Equivalencia de atributos 244
Esquema

 auxiliar 378
 de aplicación 198
 global 379
 integrado 244
 lógico 57

Esqueleto 111, 127
Estampilla de tiempo 296
Evaluación de consulta 249
Except 137
Exist 147

f

Falla
 memoria secundaria 337
 sistema 337
 transacción 337

Filtro 388

Forma normal 163
 1FN 164
 2FN 177
 3FN 179
 3FNBC 181
 4FN 185
 5FN 186

Fórmula 108, 110, 115

Fragmentación
 horizontal 363

- mixta 363
vertical 363
- From 136
Función de pertenencia 451
- g**
- Generación de bd
 primera 25
 segunda 26
 tercera 27
 cuarta 48
 quinta 49
- Generalización 68
- Grafo de dependencias 293
- Gránulo 303
- Grant 317
- Group by 136, 145
- h**
- Having 136, 145
- Herencia 68, 204
 múltiple 71, 405
- Herramientas 45
- Homonimia 232
- i**
- Identificador interno 403, 407
- Implementador de la bd 40
- Independencia
 física 43
 lógica 43
- Indicador de modificación 343
- Información 56
- Insert into values 134
- Interbloqueo 296
- Integración de
 asociaciones 237
 composiciones 242
 enlaces 236
- esquemas 199
herencias 240
- Integridad 36, 315
 de dominio 87
 de relación 88
 referencial 88
- Intersección 91
- Intersect 137
- Itinerador 291
- Itinerario 291
 consistente 292, 370
 local 369
- j**
- Join 100
 externo 103
 equi- 100
 implementación 256
 ciclos iterados 256
 hash 260
 ordenamiento y fusión 258
- natural 100
 Θ- 99, 140
- l**
- Lenguaje
 algebraico 89
 consultas 33
 definición de datos 33
 huésped 33
 manipulación de datos 33
 natural 56
 orientado a objetos 48
 predicativo 89
 relacional completo 107
- Llave 66, 177
 alterna 67, 85
 candidata 67, 85
 externa 86, 212

as 199
as 240
36, 315
inio 87
ción 88
cial 88
n 91
7
91
91
ente 292, 370
9

j

103
00
entación 256
os iterados 256
n 260
enamiento y fusión 258
100
140

i

ico 89
as 33
ón de datos 33
d 33
lación de datos 33
56
do a objetos 48
tivo 89
nal completo 107
77
67, 85
ata 67, 85
186, 212

primaria 67, 85, 207
Login 36

m

Máquina de bases de datos 385
Máquina de clientes 45
Max 144
Mediador 376
Memoria
genética 55
masiva 55
Método 402
Metodología 195
Fusion 436
Hatley 198
Hood 434
IE 198
JSD 198
Merise 198
Niam 198
Objectory 436
OMT 198, 436
OOA de Coad/Yourdon 428
OOA de Schlaer/Mellor 431
OOD de Booch 433
orientada a los datos 196
orientada a los objetos 197
orientada a los procesos 195
OORS 198
OOSD 198
OSA 198
RDD 438
SA-RT/SD-RT 198
SA/SD 198
SADT 198
Unified Method 436
Min 144
Modelaje 195
Modelo
completo 170

de datos 56, 58
entidad-asociación 57
relacional 27, 79
semántico 57

n

NF2 404
Nivel
conceptual 42
externo 42
físico 41
Normalización 161
Not exist 148

o

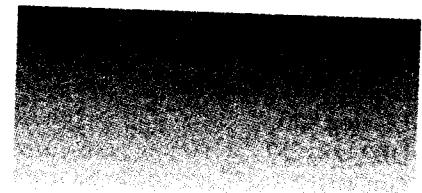
Objeto 59
Order by 146
Operación de disparo 152, 331
Operador 90
binario 90
conjuntista 90
implementación del 255
primitivo 107
unario 90
Optimización de consultas 249
Opt. consultas 129, 249, 260
Order by 136
Orientación a objetos 48, 402

p

Página 339
Password 36
Persistencia 407
Perseguidor 326
Peso de esq. de aplicación 233
Plan de ejecución 249
Polimorfismo 404
Predicado 107

- Procedimiento de acceso 249
Procesamiento de
 acceso 249
 consultas 249
Producto cartesiano 91
Programa de aplicación 34,40
Protocolo de 295
 árbol 301
 dos fases 35, 299
 expansión 299
 reducción 299
 exclusión mutua 295
 granularidad múltiple 306
 manipulación de datos 386
 validación de dos fases 372
Proyección 94
 implementación de la 254
Pseudotransitividad 171
Punto
 de control 347
 de validación 347
- q**
- Quipus 19
- r**
- Read 285
Receptor 317
Recuperación luego falla 37
Red 47, 354
 LAN 354
 MAN 354
 WAN 355
Reflexividad 170
Regla de integridad 36
Relación 80
 difusa 452
 esquema de 82
 extensión 83
- grado de 82
intensión 83
Relacional
 teoría 168
Revoke 320
Rollback 285
- s**
- Sagas 309
Saturación 172
SABD (ver SBD)
SBD 20
 deductivo 49, 442, 446
 orientado a objetos 443
 difuso 443, 451
 distribuido 356
 homogéneo 358
 heterogéneo 358, 376
 invertido 25
 jerárquico 22
 multimedios 443
 orientado a objetos 49, 409
 redes 25
 relacional 27
 relacional ampliado 408
 temporales 443, 456
- Segmento 339
Seguridad 35, 315
Selección 97
 implementación 254
 barrido 255
 índice 255
 índice secundario 255
Select 136
Semi-join 367
Sequel 128
Servidor 46
Sistema de
 archivos 30

- base de datos 20, 30
Sub-consulta 141
- t**
Tabla 130
de base 131
derivada 148
de página 342
Tabla de descomposición 175
Terabyte 39
Término 108
Transitividad 170
Transacción 35, 284
acid 287
atomicidad 287
consistencia 287
independencia 287
durabilidad 287
bien formada 288
begin 284
cuerpo de 285
de larga duración 307
- u**
Unión 91, 137, 171
externa 93
Update set where 134
Usuario final 39
- v**
Valor nulo 93
Variable libre 108
Vector
ocupación de bloques 340
situación 343
View 148
Vista 148
- w**
Where 136
WWW 380



INDICE DE AUTORES Y PRODUCTOS

a

ADDS 378
Advanced Data Servers 393
AMI 272
Amoco Production Co. 378
Amperitif Corporation 393
ANSI/SPARC 41, 127
ANSI/SQL 33
API 46
Applied Data Research 25
Armstrong W.W. 170
Atanasoff J.V. 19

b

Bachman C.W. 25
Bancilhon F. 411
Berri C. 184

c

Britton Lee, Inc. 393
Burrougs 26
Bush V. 19

CARI 378
CDC-6000 25
Chen P. 57
CICS 373
Cincom System Inc. 26
COBOL 26
CODASYL 25
Codd E.F. 27, 79, 161
Copeland G. 409
CORDS 378
Cray X-MP 392
Cullinet 26

d

Daplex 379
DATACOM/DB 25
Datalog 450
DB2 28, 257
DBC/1012 392
DBTG 25
DeMarco T. 196
DecNet 47
DECOMP 271
Delobel C. 161
Digital Equipment C. 28
DL/I 21
DMS 1100 26
DMS-II 26

f

Fagin R. 161, 184
FAP 47

g

GemStone 409, 414
General Electric 25
Gip Altaïr 411
GMIS 28
GMR 28
Gupta Corporation 28, 47

h

Hewlett-Packard 378, 409

i

IBM 257, 378
ICS/DL/I 21
IDS 25
IDMS 26
IFO 57
IMS/360 Versión 2 22
IMS/VS 22

l

Informix 28, 127
Informix Software Inc. 28, 127
Informix-SQL 131
Ingres 28, 127
INRIA 395
Iris 409, 425
IS/1 28
ISAM 21
ISO RDA 47
Itasca 419

k

King W.F. 28

m

Maier D. 409
Mega/Net 393
Microsoft/Sybase 47
MRI Systems 25
Multibase 378, 379

n

NDMS 378
NF2 404

o

O2 411
ObjectStore 417
Ontos 421
Opal 415
Oracle 28, 257
Oracle Server 47
Orion 419
OSI 47
OVD 273
OVQP 271

p

PCB 23
PDP 11/40 28

8. 127
oftware Inc. 28, 127
QL 131
127
i
25

17

k
28

m
99

93

ybase 47

ns 25

78. 379

n

o
417

57

r 47

p
8

- Perform 127
Pegasus 378
Postgres 408, 422
Postquel 422
PSB 23
- QBE 111, 127
Quel 116, 127
- R* 373
RDM/2 393
RDS* 374
RM/T 57
Rockwell International 21
RSS* 373
- SABRE 395
Server/300 393
SHM+ 57
Simula 401
Sirius-Delta 378
Smalltalk 401, 409
Software AG 25
SQL 127
SQLBase 28, 129
SQLBase Server 47
SQLMach-1 393
SQL/MP Non-Stop 258, 260
SQL Server 28, 257, 258
Square 128
Starburst 423
- Stonebraker M. 28
STRATEGY 274
Sybase 28, 257, 258
System/360 21
System-2000 25
System Development 378
System R 28, 128
- Tandem Computers 257
TCP/IP 47
Teradata Corp. 392
TOTAL 26
TRW Information Services 39
- UniSQL/M 378
UNIVAC 26
Unix 131
- VSAM 21
- Wong E. 28
- XDMS 385
XRM 28
- Zadeh L. 451
Zaniolo C. 161
Zloof M.M. 127

sistemas de Bases de Datos es una obra comprensiva sobre la tecnología de bases de datos relacionales, uno de los recursos más importantes con que se cuenta actualmente para la administración automatizada de grandes volúmenes de datos. Su autor CARLOS GONZÁLEZ ALVARADO, presenta en este libro la síntesis de su labor de más de diez años en este campo, y lo dirige a las personas interesadas en el área de bases de datos, sea como estudiantes o como profesionales.

La EDITORIAL TECNOLÓGICA DE COSTA RICA ha acogido gustosa este excelente esfuerzo en ámbito regional, para contar con un libro de bases de datos adaptado a nuestra idiosincrasia.

