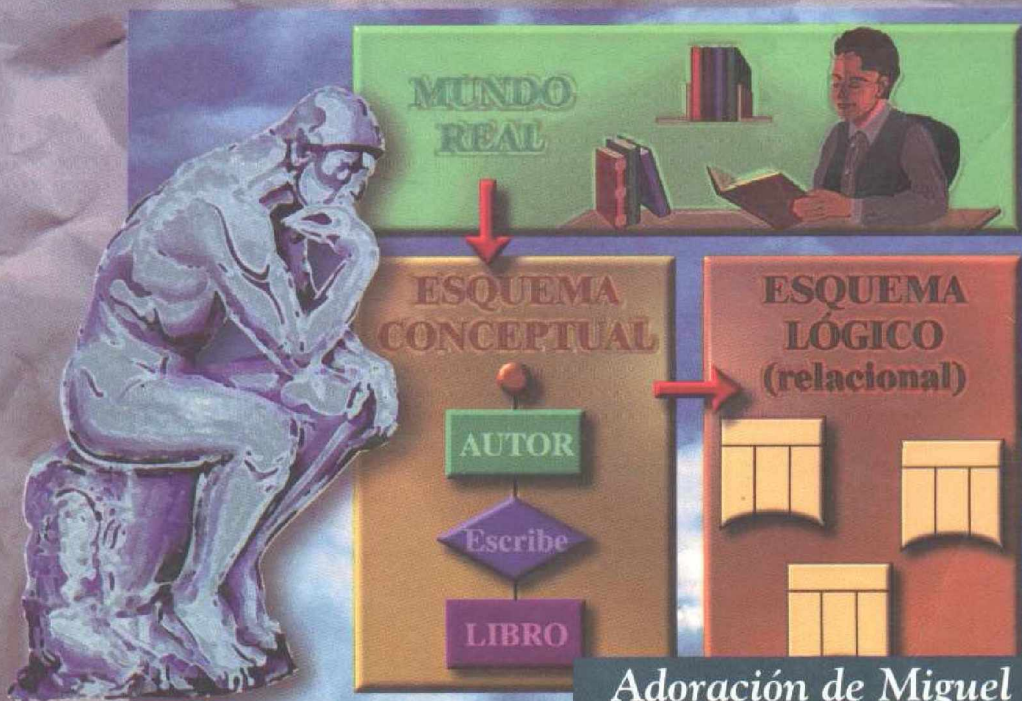


Diseño de Bases de Datos Relacionales



Adoración de Miguel
Mario Piattini
Esperanza Marcos



Diseño de bases de datos relacionales

Diseño de bases de datos relacionales

Adoración de Miguel Castaño

Universidad Carlos III de Madrid

Mario Piattini Velthuis

Universidad de Castilla-La Mancha

Esperanza Marcos Martínez

Universidad Rey Juan Carlos

Alfaomega  *ca-ma*

Diseño de bases de datos relacionales
© Adoración de Miguel Castaño, Mario Piattini Velthuis,
Esperanza Marcos Martínez

ISBN 84-7897-385-0, edición original publicada por RA-MA Editorial,
MADRID, España. Derechos reservados © RA-MA Editorial

MARCAS COMERCIALES: RA-MA ha intentado a lo largo de este libro distinguir las marcas registradas de los términos descriptivos, siguiendo el estilo de mayúsculas que utiliza el fabricante, sin intención de infringir la marca y sólo en beneficio del propietario de la misma.

© 2000 ALFAOMEGA GRUPO EDITOR, S.A. de C.V.
Pitágoras 1139, Col. Del Valle, 03100 México, D.F.

Miembro de la Cámara Nacional de la Industria Editorial
Registro No. 2317

Internet: <http://www.alfaomega.com.mx>
Email: ventas@alfaomega.com.mx
ISBN 970-15-0526-3

Derechos reservados.

Esta obra es propiedad intelectual de su autor y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

NOTA IMPORTANTE

La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro y en el disquete adjunto, ni por la utilización indebida que pudiera dársele.

Edición autorizada para venta en México y todo el continente americano

Impreso en México – Printed in Mexico

AUTORES

ADORACIÓN DE MIGUEL CASTAÑO

Licenciada en Ciencias Físicas por la Universidad Complutense de Madrid, Doctora y Licenciada en Informática por la Universidad Politécnica de Madrid. Pertenece al Cuerpo de Estadísticos Facultativos del INE. Profesora titular de la Facultad de Informática de la U.P.M., en comisión de servicios en la Universidad Carlos III de Madrid. Ha sido jefe del Servicio de Análisis y Programación, Directora de Informática y del Programa de Bancos de Datos del INE. Su labor académica se extiende a lo largo de más de treinta años en la U.C.M., Universidad Autónoma, U.P.M. y Universidad Carlos III de Madrid. Es autora de varios libros, entre los que destacan: *Derecho a la información frente al derecho a la intimidad: su incidencia en los sistemas de información estadística* y *Concepción y Diseño de Bases de Datos*, así como un centenar de artículos y comunicaciones en el área de las bases de datos, Ingeniería del Software, informática jurídica, estandarización, etc. Ha organizado, dirigido e impartido numerosos cursos, seminarios, conferencias, etc. en distintos centros de España y del extranjero, y ha dirigido o participado en diversos proyectos de investigación. Ha sido consultora de Naciones Unidas y de otros organismos internacionales (IBI, SIECA, etc.) en varios proyectos informáticos. Pertenece a diversas asociaciones científicas o profesionales nacionales y extranjeras.

MARIO GERARDO PIATTINI VELTHUIS

Doctor y Licenciado en Informática por la Universidad Politécnica de Madrid. Master en Auditoría Informática (CENEL). Especialista en la Aplicación de Tecnologías de la Información en la Gestión Empresarial (CEPADE-UPM). CISA (Certified Information Systems Auditor) por la ISACA (Information Systems Audit

and Control Association). Ha sido director del departamento de Desarrollo de Bases de Datos de SiE, S.A., socio-fundador y director de los departamentos de Formación, Metodologías e I + D de la empresa Cronos Ibérica, S.A. y director técnico de Esinet, S.L. Ha ejercido su labor académica en la Universidad Complutense de Madrid y en la Universidad Carlos III de Madrid. Ha trabajado como consultor y profesor para numerosas empresas y organismos, entre los que destacan: Siemens-Nixdorf, Ministerio de Industria y Energía, Ministerio de Interior, ATOS-ODS, UNISYS, HP, ICM, etc. Actualmente es Profesor Titular de Universidad en la Escuela Superior de Informática de la Universidad de Castilla-La Mancha. Es coautor de varios libros: *Concepción y Diseño de Bases de Datos* (Ra-Ma, 1993), *Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión* (Ra-Ma, 1996), *Auditoría Informática: Un enfoque práctico* (Ra-Ma, 1998), etc. Pertenecer a diversas asociaciones profesionales (IEEE, ACM, ISACA, PMI, AENOR, ATI, ALI, AII, OAI, AEC...)

ESPERANZA MARCOS MARTÍNEZ

Doctora y Licenciada en Informática por la Universidad Politécnica de Madrid. Diplomada en Informática por Universidad de Valladolid. Durante cinco años ha sido profesora en el Departamento de Informática de la Universidad Carlos III de Madrid, y responsable del Laboratorio de Bases de Datos Avanzadas de la Escuela Politécnica Superior. Ha trabajado en la Escuela de Informática del Centro Español de Nuevas Profesiones. Actualmente es profesora en la Escuela Superior de Ciencias Experimentales y Tecnología (ESCET) de la Universidad Rey Juan Carlos de Madrid y profesora del Máster en Ingeniería del Software y del Conocimiento de la Universidad Politécnica de Madrid. Es autora de numerosos artículos en las áreas de bases de datos, orientación a objetos, Ingeniería del Software, estandarización, etc. Ha sido representante de AENOR en los comités internacionales del ISO/IEC JTC1/SC21 WG3 DBL sobre estandarización del lenguaje SQL3. Ha organizado, dirigido e impartido numerosos cursos y seminarios en distintos centros de España sobre tecnología de bases de datos y orientación a objetos. Es coordinadora del Grupo de Objetos de la Asociación de Técnicos en Informática (ATI). Participa, también, en diversos proyectos en colaboración con empresas como Intesys, S. A., Dycsa, etc.

CONTENIDO

PRÓLOGO	XV
PREFACIO	XVII
PARTE I	1
CAPÍTULO 1. MODELO DE DATOS	3
1. INTRODUCCIÓN	3
2. MODELO, ESQUEMA Y EJEMPLAR	8
3. TIPOS DE ABSTRACCIÓN EN EL DISEÑO DE BASES DE DATOS	12
3.1. Clasificación/Particularización	13
3.2. Agregación/Desagregación	15
3.3. Generalización/Especialización	18
3.4. Asociación/Disociación	20
3.5. Jerarquías de abstracciones	21
4. CONCEPTO DE MODELO DE DATOS	23
4.1. Estática	25
4.2. Dinámica	26
5. RESTRICCIONES DE INTEGRIDAD	28
5.1. Componentes de una restricción	31
5.2. Clasificación de las restricciones	33
6. LOS MODELOS DE DATOS EN EL PROCESO DE DISEÑO DE UNA BASE DE DATOS	38
ANEXO. CLASIFICACIÓN DE LAS RESTRICCIONES	42

CAPÍTULO 2. MODELO ENTIDAD/INTERRELACIÓN (ME/R)	47
1. PRESENTACIÓN DEL MODELO	47
2. ESTÁTICA DEL MODELO E/R	49
2.1. Entidad	49
2.2. Interrelación	51
2.3. Dominio y valor	53
2.4. Atributo	54
3. RESTRICCIONES	56
4. PRIMERA APROXIMACIÓN A LA SEMÁNTICA DE LAS INTERRELACIONES	61
4.1. Elementos de un tipo de interrelación	61
4.2. Cardinalidad de un tipo de entidad	63
4.3. Atributos de las interrelaciones	65
4.4. Dependencia en existencia y en identificación	66
5. CONTROL DE REDUNDANCIA	68
5.1. Atributos derivados	68
5.2. Interrelaciones redundantes	69
6. INTERRELACIONES DE GRADO SUPERIOR A 2	72
7. OTRAS RESTRICCIONES SOBRE INTERRELACIONES	74
7.1. Restricción de Exclusividad	74
7.2. Restricción de Exclusión	76
7.3. Restricción de Inclusividad	77
7.4. Restricción de Inclusión	77
8. GENERALIZACIÓN/ESPECIALIZACIÓN	78
9. AGREGACIÓN	84
10. LA DIMENSIÓN TEMPORAL EN EL MODELO E/R	87
11. REPRESENTACIÓN GRÁFICA	89
ANEXO. SIMBOLOGÍA DEL MODELO ENTIDAD/INTERRELACIÓN ...	89

CAPÍTULO 3. MODELO DE DATOS RELACIONAL	93
1. HISTORIA Y OBJETIVOS	93
2. ELEMENTOS PERMITIDOS	95
2.1. Dominios, Relaciones y Atributos	95
3. INTENSIÓN Y EXTENSIÓN DE UNA RELACIÓN	97
4. ELEMENTOS NO PERMITIDOS: RESTRICCIONES	98
4.1. Restricciones inherentes	98
4.2. Restricciones semánticas	99
4.2.1. Clasificación de las restricciones según los elementos a los que afecta la condición	101
4.2.2. Restricciones de condición y acción específicas	103
4.2.3. Restricciones de condición general (predicado libre) y de acción específica (rechazo)	106

4.2.4. Restricciones de condición general y acción general (disparadores).....	107
5. LOS TRES NIVELES DE ANSI EN EL MODELO RELACIONAL.....	108
5.1. El nivel conceptual del modelo relacional: Esquema de Relación y Esquema Relacional.....	108
5.2. Las vistas y el nivel externo en el modelo relacional.....	109
5.3. El nivel interno en el modelo relacional.....	110
5.4. Correspondencia de la arquitectura ANSI y el modelo relacional.....	110
 PARTE II	 113
 CAPÍTULO 4. CONCEPTO Y MANIPULACIÓN DE DEPENDENCIAS FUNCIONALES	 115
1. DEPENDENCIAS ENTRE LOS DATOS.....	115
2. CONCEPTO DE DEPENDENCIA FUNCIONAL	117
2.1. Dependencia funcional plena o completa.....	120
2.2. Dependencia funcional trivial	121
2.3. Dependencia funcional elemental.....	121
2.4. Dependencia funcional transitiva	121
3. IMPLICACIÓN LÓGICA DE DEPENDENCIAS FUNCIONALES Y AXIOMAS DE ARMSTRONG.....	123
3.1. Consecuencia lógica y derivación de dependencias funcionales.....	124
3.2. Axiomas de Armstrong	125
4. DEFINICIÓN FORMAL DE SUPERCLAVE Y DE CLAVE DE UNA RELACIÓN	129
5. MANIPULACIÓN DE DEPENDENCIAS FUNCIONALES EN BASE AL CIERRE TRANSITIVO DE UN DESCRIPTOR.....	130
5.1. Cierre transitivo de un descriptor	131
5.2. Determinación de si una dependencia está implicada por un conjunto de dependencias (pertenecer a su cierre)	132
5.3. Equivalencia de dos conjuntos de dependencias	132
5.4. Recubrimiento irredundante de un conjunto de dependencias	134
5.5. Determinación de si un descriptor es clave de una relación	138
5.6. Obtención de las claves candidatas de un esquema.....	140
ANEXO. PROCEDIMIENTO DE CÁLCULO DE CLAVES	140
 CAPÍTULO 5. TEORÍA DE LA NORMALIZACIÓN: FORMAS NORMALES BASADAS EN LAS DEPENDENCIAS FUNCIONALES	 147
1. NECESIDAD DE UN MÉTODO FORMAL DE DISEÑO RELACIONAL.....	148

2.	TEORÍA FORMAL DE LA NORMALIZACIÓN DE ESQUEMAS RELACIONALES	150
2.1.	Conservación de la información.....	152
2.2.	Conservación de las dependencias	155
3.	DEFINICIÓN FORMAL DE LAS TRES PRIMERAS FORMAS NORMALES	155
3.1.	Primera forma normal (1FN).....	156
3.2.	Segunda forma normal (2FN).....	157
3.3.	Tercera forma normal (3FN).....	159
3.4.	Forma normal de Boyce-Codd (FNBC)	160
4.	DOS ENFOQUES DE DISEÑO RELACIONAL: ANÁLISIS Y SÍNTESIS	162
4.1.	Análisis.....	165
4.1.1.	Descomposición en proyecciones independientes.....	166
4.1.2.	Descomposición hasta FNBC.....	168
4.1.3.	El proceso de descomposición.....	170
4.2.	Proceso de síntesis.....	172

CAPÍTULO 6. FORMAS NORMALES AVANZADAS Y REORGANIZACIÓN DE RELACIONES 175

1.	SEMÁNTICA DE LOS DATOS Y NUEVOS TIPOS DE DEPENDENCIAS	175
2.	DEPENDENCIAS MULTIVALUADAS Y CUARTA FORMA NORMAL	177
2.1.	Dependencias multivaluadas	179
2.2.	Axiomas para la derivación de dependencias funcionales y multivaluadas	184
2.3.	Cuarta forma normal (4FN).....	185
3.	DEPENDENCIAS MULTIVALUADAS EMBEBIDAS	187
4.	DEPENDENCIAS DE COMBINACIÓN Y QUINTA FORMA NORMAL	190
4.1.	Definición de dependencia de combinación.....	193
4.2.	Quinta forma normal (5FN)	193
4.3.	Dependencia de dominio/clave.....	194
5.	OTRAS DEPENDENCIAS Y FORMAS NORMALES	194
5.1.	Dependencias de inclusión	195
6.	OTRAS CONSIDERACIONES SOBRE LA NORMALIZACIÓN DE RELACIONES.....	196
7.	REORGANIZACIÓN DE RELACIONES	198
7.1.	Estructuración de relaciones por consideraciones lógicas: particionamiento horizontal.....	200
7.2.	Reestructuración de relaciones por consideraciones de eficiencia: desnormalización y particionamiento.....	202

CAPÍTULO 7. ALGORITMOS DE DISEÑO EN EL MODELO RELACIONAL	205
1. INTRODUCCIÓN	206
2. ALGORITMOS RELATIVOS A LA NORMALIZACIÓN POR SÍNTESIS.....	208
2.1. Algoritmo de cálculo del cierre de un descriptor.....	208
2.2. Algoritmo de cálculo del recubrimiento minimal.....	209
2.3. Algoritmo de síntesis de Bernstein.....	212
2.4. Algoritmo de determinación de claves	214
3. NUEVOS ALGORITMOS DE NORMALIZACIÓN POR SÍNTESIS....	218
3.1. Algoritmo de cálculo del cierre de un descriptor.....	220
3.2. Algoritmo de cálculo del recubrimiento minimal.....	222
3.3. Algoritmo de síntesis.....	231
3.4. Algoritmo de determinación de claves	233
4. ALGORITMOS QUE DETERMINAN LA FORMA NORMAL EN LA QUE SE ENCUENTRA UN ESQUEMA DE RELACIÓN.....	238
4.1. Determinación de 2FN	238
4.2. Determinación de 3FN	242
4.3. Determinación de FNBC	245
5. ALGORITMOS DE DESCOMPOSICIÓN DE UN ESQUEMA DE RELACIÓN EN ESQUEMAS EN FNBC.....	247
5.1. Descomposición en esquemas FNBC.....	247
5.2. Proyección de un conjunto de dependencias sobre un conjunto de atributos	249
5.3. Algoritmo de descomposición en esquemas FNBC que mejora la eficiencia	251
5.4. Nuevo algoritmo de descomposición en esquemas FNBC que mejora la funcionalidad.....	255
6. DETERMINACIÓN DE SI UNA DESCOMPOSICIÓN ES SPI.....	264
6.1. Algoritmo de Ullman	265
6.2. Versión mejorada del Algoritmo de Ullman	267
7. DETERMINACIÓN DE SI UNA DESCOMPOSICIÓN PRESERVA LAS DEPENDENCIAS.....	273
8. ALGORITMO GRÁFICO PARA EL PARTICIONAMIENTO VERTICAL.....	276
8.1. Conceptos básicos	276
9. ALGUNAS CONSIDERACIONES RELATIVAS A LA EFICIENCIA Y CONCLUSIONES	285
PARTE III	287
CAPÍTULO 8. PROCESO DE CREACIÓN Y METODOLOGÍA DE DESARROLLO DE BASES DE DATOS.....	289
1. INTRODUCCIÓN AL CICLO DE VIDA DE UNA BASE DE DATOS. .	289
2. ESTUDIO PREVIO Y PLAN DE TRABAJO.....	290

2.1. Decisión política y fijación de objetivos (estudio de viabilidad).....	290
2.2. Evaluación previa de medios y costes	291
2.3. Aprobación de una estructura orgánica	291
2.4. Plan de trabajo detallado	292
3. CONCEPCIÓN DE LA BASE DE DATOS Y SELECCIÓN DEL EQUIPO	294
3.1. Concepción de la base de datos	294
3.2. Especificaciones de las necesidades de equipo físico y lógico.....	295
4. DISEÑO Y CARGA	296
4.1. Diseño lógico y físico.....	297
4.2. Carga y optimización de la base	297
5. UNA METODOLOGÍA PARA EL DESARROLLO DE BASES DE DATOS RELACIONALES	298
5.1. Concepto de metodología.....	298
5.2. Enfoque propuesto.....	301
5.3. Características de una metodología de diseño.....	309
6. ENTRADAS Y SALIDAS DEL PROCESO DE DESARROLLO.....	311

CAPÍTULO 9. MODELADO CONCEPTUAL..... 313

1. ETAPAS DEL MODELADO CONCEPTUAL.....	313
2. PASO DEL ESQUEMA PERCIBIDO AL ESQUEMA CONCEPTUAL	317
2.1. Enfoque para el análisis de requisitos	318
2.2. Creación de esquemas conceptuales a partir de especificaciones textuales	324
3. CARACTERÍSTICAS DEL ESQUEMA CONCEPTUAL	330
4. METODOLOGÍAS ASCENDENTES Y DESCENDENTES	332
5. EL PROCESO DE “INTEGRACIÓN DE VISTAS”	335
5.1. Resolución de conflictos	336
5.2. Análisis de redundancias de interrelaciones	340

CAPÍTULO 10. DISEÑO LÓGICO ESTÁNDAR..... 343

1. ETAPAS DEL DISEÑO LÓGICO	343
2. TRANSFORMACIÓN DEL ESQUEMA CONCEPTUAL AL LÓGICO ESTÁNDAR	345
3. REGLAS CONCERNIENTES AL MODELO BÁSICO.....	347
4. REGLAS CONCERNIENTES A LAS EXTENSIONES DEL MODELO E/R	358
5. GRAFO RELACIONAL	365

CAPÍTULO 11. DISEÑO LÓGICO ESPECÍFICO Y DISEÑO FÍSICO.. 367

1. DISEÑO LÓGICO ESPECÍFICO	367
2. IMPLEMENTACIÓN DE LOS PRINCIPALES CONCEPTOS DEL MODELADO RELACIONAL.....	368
2.1. Dominios.....	368
2.2. Claves primarias.....	370
2.3. Claves ajenas.....	371
2.4. Otros conceptos del modelo relacional.....	373
3. OBJETIVOS Y ACTIVIDADES DEL DISEÑO FÍSICO.....	373
4. CONCEPTOS GENERALES DE ALMACENAMIENTO DE LOS DATOS EN SOPORTE SECUNDARIO.....	376
4.1. Diseño de bloques y gestión de almacenamiento intermedio.....	377
5. ORGANIZACIÓN DE ARCHIVOS Y MÉTODOS DE ACCESO	379
5.1. Organizaciones consecutivas.....	380
5.2. Organizaciones direccionadas	381
5.3. Organizaciones indizadas	384
6. OTRAS TÉCNICAS DE DISEÑO FÍSICO.....	388
6.1. Agrupamientos (cluster) de tablas.....	388
6.2. Técnicas de compresión	390
6.3. Redundancia de datos.....	390

PARTE IV 391

CAPÍTULO 12. HERRAMIENTAS DE DESARROLLO: LENGUAJES DE CUARTA GENERACIÓN..... 393

1. EVOLUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN.....	393
1.1. Lenguajes de primera y segunda generación.....	393
1.2. Lenguajes de tercera generación	394
1.3. Lenguajes de cuarta generación	395
1.4. Lenguajes de quinta generación	395
1.5. Lenguajes orientados a objetos.....	396
1.6. Lenguajes “visuales”	396
2. COMPONENTES DE UN L4G.....	397
3. VENTAJAS E INCONVENIENTES DE LOS L4G	398
3.1. Ventajas.....	398
3.2. Inconvenientes.....	399

CAPÍTULO 13. SISTEMAS DE DICCIONARIOS DE RECURSOS DE INFORMACIÓN..... 401

1. INTRODUCCIÓN.....	401
----------------------	-----

2.	EVOLUCIÓN HISTÓRICA: DE LOS DIRECTORIOS/DICCIONARIOS DE DATOS AL DICCIONARIO DE RECURSOS DE INFORMACIÓN	402
3.	EL SDRI Y SU ENTORNO.....	406
4.	PAPEL DEL DICCIONARIO EN LA EMPRESA	407
5.	CONTENIDO DEL DICCIONARIO DE RECURSOS DE INFORMACIÓN	408
6.	ESTÁNDARES SOBRE SDRI.....	409
7.	ESTÁNDARES ISO/IEC PARA SDRI.....	411
	7.1. Marco de referencia de SDRI.....	411
	7.2. Interfaz de servicios	414
	7.3. Otros estándares de ISO/IEC.....	414

CAPÍTULO 14. HERRAMIENTAS CASE Y DISEÑO DE BASES DE DATOS 417

1.	INTRODUCCIÓN.....	417
2.	CATEGORÍAS DE HERRAMIENTAS CASE.....	420
3.	HERRAMIENTAS DE DISEÑO DE BASES DE DATOS	421
	3.1. Clases de herramientas	421
	3.2. Deficiencias de la tecnología CASE para el diseño de bases de datos	422
	3.3. Resumen de algunos proyectos y herramientas de desarrollo de bases de datos	423
4.	MARCO PARA LA EVALUACIÓN DE HERRAMIENTAS DE DISEÑO DE BASES DE DATOS	427
5.	ENEAS/BD: UN ENTORNO PARA LA ENSEÑANZA AVANZADA DE SISTEMAS DE BASES DE DATOS.....	428

APÉNDICES 433

A.	EJERCICIOS PROPUESTOS	435
B.	EJEMPLO COMPLETO.....	479
C.	MANUAL DE USUARIO DE LA HERRAMIENTA RENO V.3	499
D.	LISTA DE ACRÓNIMOS Y ABREVIATURAS.....	529

BIBLIOGRAFÍA 535

ÍNDICE ALFABÉTICO 547

PRÓLOGO

Los motivos para prologar un libro pueden ser diversos, tanto por parte de los autores, que lo piden, como del que lo realiza. En mi caso me animan dos motivos: el primero de ellos es la amistad y el segundo la calidad de la obra.

En una Universidad barrida muchas veces por el Despotismo escudado en la falta de autoridad, de liderazgo intelectual y de órganos de decisión que los respalden, la Amistad es un valor escaso. En mi caso debo remontarme a mediados de los sesenta, momento en el que conocí a Adoración de Miguel para fijar el inicio de la nuestra: eran aquellos tiempos de comienzo de la Informática en España, en Madrid. A la solidez profesional Adoración añade la virtud de la coherencia, de la tenacidad profesional. Para aquellos que miramos ahora Madrid desde el Mediterráneo, la mayor parte de las actividades de I+D en Bases de Datos han tenido allí la autoría de Adoración.

Adoración ha producido además discípulos: Mario Piattini y Esperanza Marcos (coautores)... que se han formado en esa Escuela que sólo los verdaderos profesores son capaces de formar. Para ello hace falta continuidad en el esfuerzo, en el interés, en la exigencia, en la generosidad. Y éste es el caso.

Con Mario Piattini mi relación es más reciente; pero, la actividad científica la hacen personas que normalmente no vienen de la nada. Mario está entroncado, vía su padre, con una institución que en los confusos momentos iniciales de la Informática en España desempeñó un papel importante: el IBI (localizado en Roma) en el que éste trabajó.

Como a veces digo, Mario es la persona que me ha contado METRICA 3 con detalle ¡sin que me aburra! A sus grandes dotes como profesor une una sólida experiencia en la empresa y una enorme capacidad intelectual y de trabajo.

Esperanza Marcos irrumpió con fuerza en el escenario durante la celebración del congreso EDBT 98 en Valencia durante marzo de 1998. A su incesante actividad en las JIDBD 97, de las que fue su hacedora, añadía su bien hacer profesional.

Y éstos son para mí, los autores y mi relación de amistad con ellos.

Y ¿el libro?

El libro constituye un buen producto en el que la precisión de los conceptos se conjuga con una prosa fluida y amena. Los ejemplos proliferan a lo largo de la exposición y la claridad se une a la naturalidad. Es un libro sobre Diseño de Bases de Datos para ser leído agradablemente por un alumno de Ingeniería Informática, por un profesional de las Bases de Datos, por un investigador que recurre a las fuentes... en estos momentos en los que la lectura cede terreno ante nuevas formas de comunicación.

No es un libro plagado de formalismos, pero sí de conceptos precisos, profesionalmente adquiridos, útiles de inmediato. Es un libro de ahora.

Por los motivos expuestos he aceptado prologar este libro. He de añadir además otro: mi compromiso irrenunciable con el buen hacer en Informática, venga de donde venga.

En este caso además viene de tres amigos.

Isidro Ramos
Valencia, junio de 1999

PREFACIO

La creación de un Sistema de Información abarca dos grandes áreas claramente diferenciadas, aunque fuertemente relacionadas: los datos y los tratamientos. Si bien la concepción y el diseño del sistema de datos y la del conjunto de tratamientos no puede realizarse de forma independiente, los problemas a resolver son de naturaleza distinta y nuestro objetivo en esta obra se centra en los datos.

Sin embargo, y a pesar de sus más de tres décadas de existencia, de sus miles de usuarios en el mundo entero, y de la extraordinaria atención que han dedicado al tema científicos y técnicos de reconocida valía, la concepción y diseño de bases de datos sigue siendo una tarea larga, difícil y costosa que no debe improvisarse, ya que lleva consigo una serie de actividades de decisión y planificación muy complejas y variadas. Estas dificultades inherentes al diseño de una base de datos han de tener una adecuada respuesta metodológica.

En este libro hemos intentado conjugar aspectos teóricos y prácticos, poniendo al alcance de los lectores nuestra experiencia en la aplicación, investigación y docencia en el área de las bases de datos, procurando, como decía Einstein: *“hacerlo tan sencillo como sea posible, pero no más”*.

La obra se centra en el diseño de bases de datos relacionales; y éstos son los objetivos que nos hemos propuesto al escribirla:

- Presentar de forma clara y precisa el concepto de modelo de datos.
- Enfatizar la importancia de un modelado conceptual semántico, al más alto nivel, utilizando el modelo E/R extendido.

- Proporcionar unos principios metodológicos que ayuden a realizar un buen diseño conceptual y a llevar a cabo la transformación del esquema conceptual obtenido a un esquema lógico con la mínima pérdida de semántica.
- Suministrar una sólida base teórica, como es la teoría de la normalización, al diseño lógico de bases de datos, permitiendo así aplicar procedimientos algorítmicos a dicho diseño.
- Dar a conocer el soporte que pueden ofrecer las herramientas CASE y los diccionarios de recursos de información al desarrollo de bases de datos.

CONTENIDO

Existen cuatro partes claramente diferenciadas:

PARTE I. MODELOS DE DATOS

En la primera parte, que consta de tres capítulos, se expone, en primer lugar, el concepto de modelo de datos, analizando con detenimiento las distintas restricciones, inherentes y de usuario.

El capítulo 2 se dedica a presentar, en profundidad, el modelo Entidad/Interrelación extendido (ME/R) que sirve de base para el modelado conceptual y que se encuentra soportado por la mayoría de las herramientas CASE.

Esta parte finaliza con un capítulo que resume los principales conceptos del modelo relacional.

PARTE II. DISEÑO EN EL MODELO RELACIONAL

Esta parte analiza el diseño de bases de datos en el modelo relacional, dedicando el capítulo 4 a estudiar el concepto y la manipulación de dependencias funcionales.

Los dos capítulos siguientes presentan las formas normales basadas en dependencias funcionales (1FN, 2FN, 3FN y FNBC), y se exponen otro tipo de dependencias (multivaluadas, de combinación, de inclusión, etc.) así como las formas normales a las que dan lugar (4FN y 5FN).

El capítulo 7 presenta los principales algoritmos de normalización, recogiendo mejoras respecto a los publicados en otras obras anteriores.

PARTE III. METODOLOGÍA PARA EL DISEÑO DE BASES DE DATOS

Esta parte proporciona unos principios metodológicos que constituyen el fundamento para conseguir un diseño adecuado de una base de datos. Aunque se insiste muchas veces en que el diseño de bases de datos sigue teniendo algo de “arte”, esto no impide que sea imprescindible conocer un conjunto de conceptos y técnicas que ayuden a conseguir un buen diseño.

En el capítulo 8 se presenta el proceso global de creación de una base de datos, así como las características que debe poseer una metodología de diseño de bases de datos.

El modelado conceptual se aborda en el siguiente capítulo, que incluye la descripción de la técnica de integración de vistas.

El capítulo 10 presenta las reglas que permiten transformar un esquema conceptual (E/R) en un esquema lógico (relacional).

Por último, se exponen, de forma resumida, algunos conceptos sobre diseño lógico específico y diseño físico de bases de datos.

PARTE IV. HERRAMIENTAS

En esta parte se resumen las principales herramientas relacionadas con el desarrollo de bases de datos, comenzando por los lenguajes y entornos de cuarta generación, que se abordan en el capítulo 12. En el capítulo siguiente se analizan los sistemas de diccionario de recursos de información que, a nuestro juicio, constituyen una de las piezas claves en la arquitectura de un sistema de información.

El capítulo 14 se dedica al estudio del soporte que ofrecen las herramientas de ayuda al diseño (CASE) en el proceso de creación de una base de datos.

Por último se incluyen algunos apéndices, donde se presentan distintos ejercicios, varios de ellos resueltos, un ejemplo completo desarrollado en SQL Base y en ORACLE (en este caso con la ayuda de DESIGNER) y el manual de RENO, una herramienta para la normalización de relaciones, incluida en un disquete en esta obra.

Hay que destacar la gran cantidad de ejercicios que se recogen: comprenden problemas de normalización y diseño de bases de datos aplicando la metodología expuesta en esta obra.

ORIENTACIÓN A LOS LECTORES

El libro se dirige a una audiencia muy variada que abarca, entre otros, a:

A) Alumnos de la asignatura de Diseño de Bases de Datos en facultades o escuelas universitarias

Puede servir de libro de texto para una asignatura cuatrimestral de Diseño de Bases de Datos o como segunda parte de una asignatura anual de Bases de Datos. En este caso se debería estudiar todo el material del libro, pudiéndose seguir el orden del mismo o modificándolo de la siguiente manera.

Se empezaría por el capítulo 1 para sentar las bases sobre el concepto de modelo de datos, que se plasmarían de forma práctica en el modelo E/R (capítulo 2). De esta manera, en caso de que se impartiera en paralelo la asignatura de Ingeniería del Software, en ésta se podría abordar la técnica de diagramas de flujo de datos y así, ofrecer entre las dos asignaturas una visión completa de la fase de análisis de un sistema de información. Podría ser conveniente en este caso presentar también los contenidos del capítulo 14, para que los alumnos realizaran prácticas de modelado conceptual utilizando alguna herramienta CASE.

A continuación se tratarían los capítulos 8 y 9, que resumen el proceso de creación de una base de datos y que profundizan en el modelado conceptual, a partir de entrevistas con el usuario, análisis de impresos, etc.

Posteriormente, se puede iniciar la fase de diseño lógico, empezando para ello con un repaso del modelo relacional (capítulo 3) y explicando las reglas de transformación del esquema conceptual al esquema lógico (capítulo 10).

A continuación, se podría abordar la teoría de la normalización, capítulos 4 al 7, en principio de una manera algo más intuitiva, para ser formalizada en un segundo momento.

Una vez acabada esta teoría, se pueden presentar algunos aspectos de diseño físico que se encuentran íntimamente relacionados con la reestructuración de relaciones.

B) Alumnos del módulo profesional 4: desarrollo de aplicaciones en entornos de cuarta generación y con herramientas CASE del ciclo formativo de grado superior correspondiente al título de Técnico Superior en Desarrollo de Aplicaciones Informáticas

Esta obra, junto con la anterior de *Modelos y Fundamentos de Bases de Datos*, cubre todo el contenido de este módulo, que según establece el Real Decreto 1676/1994, tiene una duración de 310 horas.

Recomendaríamos como texto básico para el alumno el libro anterior, mientras que éste se encuentra más dirigido al profesor, ya que amplía algunos contenidos, tratando

temas (como las herramientas) que se pueden exponer acompañados de productos comerciales de los que disponga el centro.

La gran cantidad de ejercicios resueltos creemos que facilitará la labor del profesor en la impartición de este módulo.

C) Profesionales informáticos que estén trabajando en el área de bases de datos

Este tipo de personas (analistas, programadores, etc.) muchas veces tienen ya conocimientos prácticos bastante profundos sobre productos concretos, que podrían ser perfectamente completados con el rigor teórico y la estructuración de conocimientos que le propone este libro.

Estos lectores pueden consultar los temas de manera independiente, y creemos que les resultará muy útil, sobre todo, el capítulo 2 sobre modelado conceptual y la parte III que expone de manera sistemática el proceso de desarrollo de una base de datos.

OTRAS OBRAS RELACIONADAS

Los lectores interesados en los contenidos de este libro pueden considerar útil otras obras relacionadas que hemos publicado en la editorial RA-MA:

DE MIGUEL, A. y PIATTINI, M. (1999). *Fundamentos y modelos de bases de datos*. 2.^a ed. Ed. Ra-Ma, Madrid.

Este libro es el complemento ideal a la presente obra, ya que presenta los conceptos fundamentales de la tecnología de bases de datos, analizando en profundidad el modelo relacional y el lenguaje SQL.

PIATTINI, M., CALVO-MANZANO, J. A., CERVERA, J. y FERNÁNDEZ, L. (1996). *Análisis y diseño detallado de Aplicaciones Informáticas de Gestión*. Ed. Ra-Ma, Madrid.

En esta obra se presentan los principios del análisis y diseño de Sistemas de Información, profundizando en las técnicas de desarrollo de funciones, que complementan las técnicas de diseño de datos expuestas en la presente obra. También se abordan temas relativos a la calidad, pruebas, verificación y validación, gestión de proyectos, mantenimiento, reingeniería y herramientas CASE.

TESTIMONIO DE RECONOCIMIENTO

Queremos agradecer, en primer lugar, a Paloma Martínez y a Dolores Cuadra, profesoras de la Universidad Carlos III de Madrid, sus aportaciones en los capítulos de modelado conceptual y diseño físico, respectivamente. También a Visitación López, Julia Martínez y Henar Pinilla, cuyos Proyectos de Fin de Carrera dirigidos por Adoración de Miguel son la base del capítulo 7 y de la herramienta RENO que se incluye en un disquete.

A Juan Canela, que no sólo nos ha ayudado en la siempre difícil e ingrata labor de preparación del original y elaboración de figuras que supone la creación de una obra como la presente, sino que también, sus amplios conocimientos de bases de datos especialmente de diseño, le han permitido hacernos interesantes sugerencias y comentarios, colaborando en la realización de muchos de los ejemplos. A los profesores Elena Castro y Carlos Nieto, que junto con las dos profesoras anteriormente citadas y el becario de FPI José M^a Cavero forman el grupo de bases de datos de la Universidad Carlos III de Madrid, así como a todos los integrantes del Laboratorio de Bases de Datos Avanzadas de dicha Universidad, especialmente a Gregorio Celada, Roberto Hens, Belén Vela, Ana Belén Parrilla, Isabel Rodríguez, etc. Todos ellos se han hecho acreedores de nuestro reconocimiento por su ayuda en la consecución de esta obra

También deseamos agradecer a los numerosos alumnos de la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, la Facultad de Informática de la Universidad Politécnica de Madrid, la Facultad de Matemáticas y la Escuela Superior de Informática de la Universidad Complutense de Madrid, y la Escuela Superior de Informática de la Universidad de Castilla-La Mancha, así como de la Escuela Superior de Ciencias Experimentales y Tecnología de la Universidad Rey Juan Carlos por sus valiosos comentarios

Parte del material de esta obra se ha desarrollado en el marco del proyecto TIC96-0753 subvencionado por la CICYT. Deseamos, por esta razón, mostrar nuestro agradecimiento a la CICYT por la subvención concedida, lo que nos ha permitido dotar a nuestro laboratorio del equipo necesario para instalar diversas herramientas de bases de datos.

Nuestro reconocimiento a los directores y consejos de redacción de las revistas: *ALGORITMO*, *ALIBASE*, *CHIP*, *COMPUTERWORLD*, *CUORE*, *NOVATICA* y *SIC*, por animarnos a publicar artículos sobre la tecnología de las bases de datos y otros temas relacionados que han servido de base para algunos capítulos del libro.

Adoración de Miguel desea hacer constar que han sido muchas las personas cuya ayuda, ánimo y apoyo a lo largo de los años han colaborado a que este libro sea una realidad; en especial, los numerosos alumnos participantes en cursos y seminarios que ha impartido tanto en España como en diferentes países de Hispanoamérica, así como en

la actualidad sus compañeros, profesores del equipo de Bases de Datos Avanzadas (ya citados) y del Grupo SINTONÍA (recientemente creado) de Investigación en Sistemas de Información e Ingeniería del Software, en especial a Antonio de Amescua, Ángel García, Juan Lloréns, Paloma Domingo y Belén Ruiz. En todos ellos ha pensado en el momento de escribir estas páginas.

Mario Piattini quiere agradecer a la Universidad de Castilla-La Mancha el apoyo recibido para la realización de esta obra, así como para sus tareas docentes e investigadoras; especialmente a los miembros del grupo ALARCOS: Francisco Ruiz, Macario Polo, Coral Calero, Marcela Genero, Eduardo Fernández-Medina, Antonio Martínez y Manuel Serrano.

Esperanza Marcos agradece tanto a la Universidad Carlos III de Madrid como a la Universidad Rey Juan Carlos, en las que trabajaba durante la realización de esta obra, por el apoyo recibido, y al grupo de software de la URJC, especialmente a Ángel Velázquez, Margarita Martínez, Sascha Ossowski, Mercedes de la Cámara, Javier Sáenz y Carlos Sobrino. Así mismo, quiere dar las gracias a Adoración de Miguel y a Mario Piattini por las enseñanzas recibidas de ellos a lo largo de estos años.

El libro ha sido amablemente prologado por el Profesor Isidro Ramos de la Universidad Politécnica de Valencia, al cual deseamos agradecer no sólo este prólogo, sino también su esfuerzo y labor a lo largo de muchos años a favor de la Informática.

Por último, nos resta expresar nuestro reconocimiento a Ana M.^a Reyes por sus valiosas sugerencias que han contribuido a mejorar notablemente este libro, así como a la empresa ALBADALEJO, S. L., que se encargó de la maquetación del mismo, y a la editorial Ra-Ma, especialmente a José Luis Ramírez, por su continuo apoyo y colaboración.

Adoración de Miguel
Mario Piattini
Esperanza Marcos

Madrid, octubre 1999

- 1. Modelo de datos**
- 2. Modelo Entidad/Interrelación
(ME/R)**
- 3. Modelo de datos relacional**

*P
A
R
T
E
I*

CAPÍTULO 1

MODELO DE DATOS

En este capítulo se analizan los modelos de datos como herramientas de abstracción que permiten representar la realidad, captando su semántica. Discutimos primero el concepto de modelo y de esquema, para pasar a presentar diferentes tipos de abstracciones que se utilizan en el modelado de datos y a definir la estática y la dinámica de los modelos de datos. Se estudian en profundidad las restricciones semánticas, para terminar viendo el papel que desempeñan los modelos de datos en el diseño de una base de datos.

1. INTRODUCCIÓN

Desde tiempos remotos, los datos han sido registrados por el hombre en algún tipo de soporte (papel, piedra, madera, etc.) a fin de que quedara constancia de un fenómeno o idea. Los **datos** han de ser interpretados (incorporándolos significado) para que se conviertan en **información** útil. Cuando utilizamos el lenguaje natural y decimos, por ejemplo, que una persona ha nacido en 1965, el dato (1965) va acompañado de su interpretación (año de nacimiento de una cierta persona); sin embargo, en la informática, desde sus inicios, se separó el dato de su significado. Por ello, a fin de facilitar la interpretación de los datos, surgen los modelos de datos como instrumentos que ayudan a incorporar significado a los datos.

Según FLORY (1982), “modelar consiste en definir un mundo abstracto y teórico tal que las conclusiones que se puedan sacar de él coincidan con las manifestaciones aparentes del mundo real”. Siendo un *modelo*, “un conjunto de conceptos que permite construir una representación organizacional de la empresa”. Como señalan TSICHRITZIS y LOCHOVSKY (1982), “un modelo de datos es un dispositivo de

abstracción que nos permite ver el bosque (esto es, la información contenida en los datos) en oposición a los árboles (valores individuales de los datos)”.

Según el DRAE¹, la *abstracción* es la acción y el efecto de abstraer, “separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción”. Por tanto, la abstracción, como proceso mental capaz de ocultar detalles y fijarse en lo esencial, busca las propiedades comunes de un conjunto de objetos, reduciendo así la complejidad y ayudando a la comprensión del mundo real.

Los modelos de datos proporcionan mecanismos de abstracción que permiten la representación de aquella parcela del mundo real cuyos datos nos interesa registrar, lo que habitualmente se denomina *universo del discurso* o, en palabras de DITTRICH (1994) *mini-mundo*. Dicha representación se concibe en dos niveles: el de las **estructuras** que hacen posible la representación de la información, y el de la **información** en sí misma. Estos dos niveles dan lugar, en el ámbito de las bases de datos, a la distinción entre **esquema** y **base de datos**; conceptos que DITTRICH (1994) define como: “*La descripción específica de un mini-mundo determinado, en términos de un modelo de datos, recibe el nombre de **esquema** (esquema de datos o esquema de base de datos) de dicho mini-mundo. La colección de datos que en sí misma representa la información del mini-mundo da lugar a la **base de datos***”.

Asociados a los modelos de datos están los lenguajes de datos que permiten definir y manipular (consultar y actualizar) la base de datos. En lo que respecta a la relación entre los modelos y los lenguajes de datos, hay que destacar que los modelos son la base para los lenguajes, aunque el nivel de abstracción de estos últimos es menor, ya que el lenguaje es el modelo más una sintaxis. La existencia de distintos lenguajes puede proceder tanto del modelo como de la sintaxis; por ejemplo, el lenguaje SQL es el resultado de aplicar una determinada sintaxis al modelo relacional, mientras que el QUEL es otro lenguaje relacional ya que la sintaxis es distinta aunque el modelo sea el mismo; el OQL es el resultado de asociar a otro modelo (el modelo de objetos –MO–) una cierta sintaxis (ver figura 1.1).

En la arquitectura de una base de datos propuesta por ANSI² –ANSI (1975) y (1978)– se suelen diferenciar tres niveles de abstracción: **Global**³, **Externo** e **Interno**. El nivel global contiene una representación del conjunto de los datos de una organización; en el nivel externo, los datos (en general, sólo una parte de los mismos) se describen para atender las necesidades de uno o varios procesos o de un grupo de usuarios en particular; el nivel interno describe las características de los datos tal como han de encontrarse almacenados físicamente, siendo sus elementos de descripción punteros, índices, agrupamientos, etc.

¹ Diccionario de la Real Academia Española. Vigésima primera edición (1992).

² ANSI es el acrónimo de American National Standard Institute, es decir, la organización oficial de estándares de Estados Unidos.

³ Para ANSI, **Conceptual**; posteriormente haremos la distinción entre global y conceptual.

LD = MD + Sintaxis

Ejemplos:

SQL = MDR + Sintaxis

QUEL = MDR + Sintaxis (distinta)

OQL = MO + Sintaxis

Figura 1.1. Modelos de datos y lenguajes de datos

Existen, por tanto, en una base de datos tres clases de esquemas: el **esquema global**, los **esquemas externos** (tantos como necesiten las aplicaciones), y el **esquema interno** que, en un momento determinado, es único, aunque un mismo esquema global admite distintos esquemas internos entre los cuales se seleccionará aquel que cumpla mejor los requisitos de eficiencia, seguridad, etc.⁴; entre estos tres tipos de esquemas existen dos tipos de funciones de correspondencia (*mapping*), la que permite la transformación esquema global/esquemas externos y la que realiza la transformación esquema global/esquema interno; el Sistema de Gestión de la Base de Datos (SGBD) ha de proporcionar estas funciones de correspondencia. En la figura 1.2 pueden verse los tres tipos de esquemas: Esquemas Externos –EE–, Esquema Global –EG– y Esquema Interno –EI–, el cual puede variar según se va “afinando” la base de datos, pero es único en un momento determinado –EI_k–; también se han presentado en la figura las dos funciones de correspondencia.

Según el nivel de abstracción de la arquitectura a tres niveles en el que se encuentre la estructura descrita, el modelo que permite su descripción será un **modelo global, externo o interno**. De entre los distintos tipos de modelos, son los globales en los que vamos a centrar nuestra atención, ya que los externos suelen utilizar conceptos parecidos a los de los correspondientes globales y los internos, aunque tienen características comunes, realmente no existen como tales modelos ya que son propios de cada producto comercial.

A veces se utiliza la expresión **modelo lógico** para hacer referencia tanto a los modelos globales como externos, ya que ambos describen aspectos lógicos de los

⁴ El esquema interno se va afinando (*tuning*) a fin de conseguir un mejor rendimiento global de las aplicaciones, y más especialmente de las aplicaciones críticas.

datos, en contraposición a los aspectos más cercanos a la máquina que se contemplan en los modelos internos; en ocasiones, a los modelos lógicos se los denomina simplemente modelos de datos. Nosotros utilizaremos la expresión *modelo de datos* para referirnos, en general, a cualquier tipo de modelo en el campo de las bases de datos.

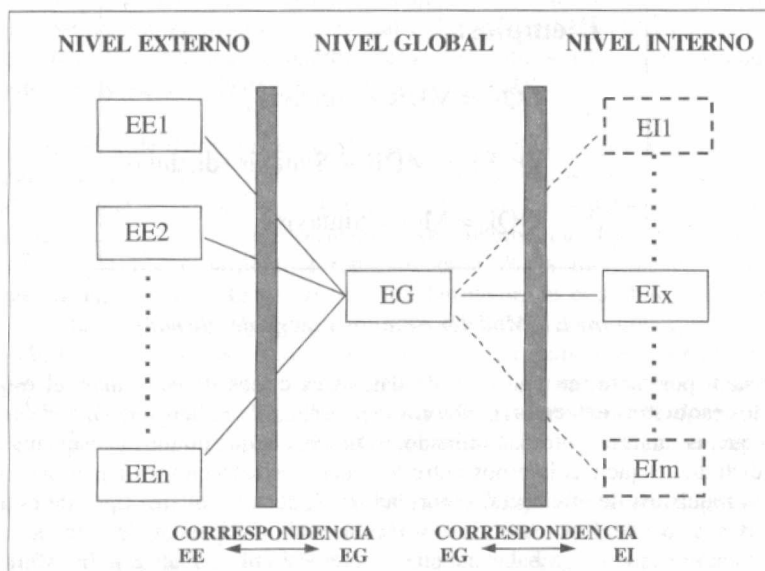


Figura 1.2. Los tres niveles de abstracción de la arquitectura ANSI

En la figura 1.3 se puede ver un ejemplo que describe una pequeña parte de una base de datos para la gestión de los cursos de doctorado de una universidad, donde aparece el esquema global, el esquema interno y dos esquemas externos que describen los datos para dos aplicaciones. En el esquema global tenemos tres tipos de objeto⁵: CURSO, PROFESOR e IMPARTE, que se transforman en registros almacenados en el esquema interno; los dos esquemas externos (uno en SQLForms y otro en Pascal) describen sólo una parte del esquema, aquella que necesitan las correspondientes aplicaciones.

⁵ El término objeto no tiene aquí el significado específico que se le atribuye en la orientación al objeto, sino la acepción del lenguaje común.

a) ESQUEMA GLOBAL	b) ESQUEMA INTERNO	c) ESQUEMAS EXTERNOS
CURSO DE DOCTORADO <i>/* Tipo de Objeto */</i>	CURSOS DE DOCTORADO <i>/* Registro Almacenado*/</i>	
<u>CURSO</u>	<u>CURSO</u>	<u>EN ORACLE FORMS</u>
CODIGO Caracter (5)	COD_CURSO Byte (3)	(listado de cursos)
NOMBRE Caracter(50)	NOMBRE Byte (50)	CODIGO Varchar2 (5)
NUM_HORAS Numérico (3)	NUM_HORAS Byte (2)	NOMBRE Varchar2 (50)
DESCRIPCION Carácter variable (200)	DESCRIPCION Byte (200)	HORAS Number (3,0)
Clave CODIGO	Índice de 2 niveles sobre COD_CURSO	DESCRIPCION Varchar2 (200)
<u>PROFESOR</u>	<u>PROFESOR</u>	<u>EN PASCAL</u>
CODIGO Caracter (3)	COD_PROFE Byte (2)	(asignación cursos a profesores)
NOMBRE Caracter(30)	NOMBRE Byte (30)	CURSO Char (5)
DNI Caracter (10)	DNI Byte (10)	NOMBRE Char (30)
DIRECCION Caracter(50)	DIRECCION Byte (50)	HORAS Integer(10)
SALARIO Numérico (7)	SALARIO Byte (4)	COD_PROFE Char (3)
Clave CODIGO	Índice 1 nivel sobre COD_PROFE	PROFESOR Char (30)
<u>IMPORTE</u>	<u>IMPORTE</u>	INICIO String(10)
PROFESOR Caracter (3)	FECHA_INI Byte (8)	FIN String(10)
CURSO Caracter (5)	FECHA_FIN Byte (8)	
FECHA INICIO Fecha	PUNTERO_CURSO Byte (4)	
FECHA FINAL Fecha	PUNTERO_PROFESOR Byte (4)	
Clave PROFESOR, CURSO		

Figura 1.3. Ejemplo de esquema global, esquema interno y dos esquemas externos para una base de datos que describe los cursos de doctorado de una universidad

Los modelos globales se clasifican, a su vez, en conceptuales y convencionales. Los **modelos conceptuales** (también denominados de alto nivel) facilitan la descripción global del conjunto de información de la empresa al nivel más próximo al usuario, por lo que sus conceptos son cercanos al mundo real (entidades, atributos, interrelaciones, etc.); los **modelos convencionales** se encuentran instrumentados en los SGBD y están orientados a describir los datos a nivel lógico para el SGBD (de ahí que también reciban el nombre de modelos lógicos o modelos de bases de datos⁶), por lo que sus conceptos son tablas o relaciones en el caso del modelo relacional, redes en el Codasyl, jerarquías en el jerárquico, etc. En la figura 1.4 se resume esta clasificación de los modelos globales.

⁶ Como podemos apreciar, la terminología no es única y esta escasa precisión terminológica es la causa muchas veces de la confusión de los usuarios, en especial de los que se están iniciando en el estudio de las bases de datos. En PASCAL (1993), apéndice 7A (pgs. 135 a 137) se critica este confucionismo terminológico y, aunque nosotros no estamos totalmente de acuerdo con algunas de sus aseveraciones, recomendamos su lectura así como hacer un estudio comparativo de su punto de vista con el que nosotros proponemos.

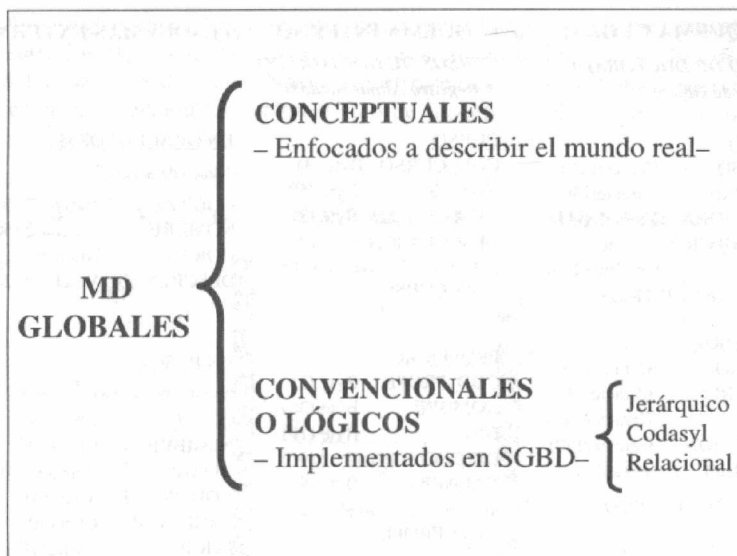


Figura 1.4. Clasificación de los modelos de datos globales

2. MODELO, ESQUEMA Y EJEMPLAR

Para algunos autores, como DITTRICH (1994), la expresión *modelo de datos* no está bien elegida para hacer referencia al concepto que representa en el ámbito de las bases de datos, ya que “... *es menos un modelo en sí mismo, que un marco para concebir modelos (del mundo real)*”⁷.

Puede que esto haya llevado en el campo de la Ingeniería del software, especialmente en las metodologías de desarrollo de sistemas de información y en su práctica, a llamar modelo de datos tanto al instrumento de descripción (lo que realmente es para nosotros el modelo) como al resultado de la misma (para nosotros esquema), sobrecargando así la expresión modelo de datos y aumentando con ello el confucionismo al que acabamos de aludir; la falta de distinción entre modelo y esquema es, en nuestra opinión, bastante perniciosa⁸.

⁷ En la tesis MARCOS (1997), se puede encontrar una interesante discusión sobre la expresión modelo de datos, llegándose a la conclusión de que, a pesar de las consideraciones de DITTRICH, sí es adecuado utilizar dicha expresión con el sentido que se le da en las bases de datos.

⁸ En la metodología MERISE se habla de *formalismos*, en lugar de modelos, como marcos para definir esquemas, ROCHFELD (1992). En Métrica versión 2.1 y en la propuesta de un grupo de investigadores de la Universidad Carlos III de Madrid, en el marco del proyecto CICYT(TIC960753) para una nueva versión de Métrica, se mantiene la distinción entre modelo de datos y esquema.

Esta ambigüedad se puede apreciar en expresiones como: “el modelo Entidad/Interrelación (E/R) de una universidad (para referirse al esquema de una universidad descrito en el modelo E/R) o “el modelo E/R” (para hacer referencia al modelo de datos E/R)⁹. Si no evitamos el doble significado del término, o si, al menos, no somos conscientes de ello, esta confusión terminológica puede dar lugar a otro tipo de ambigüedades conceptuales mucho más graves.

Consideramos que el término esquema es muy apropiado en el sentido que aquí se ha dado (y que es el habitual en el campo de las bases de datos), lo que podemos confirmar si acudimos al DRAE que define dicho término como “una representación gráfica y simbólica de una cosa atendiendo sólo a sus líneas o caracteres más significativos”. Nosotros, por tanto, distinguiremos entre estos dos conceptos: **modelo** y **esquema**, aplicándolos tal como acabamos de exponer y aparece en la figura 1.5, donde se puede observar que la descripción de un cierto mundo real (por ejemplo una universidad) por medio de un modelo de datos da como resultado un esquema.

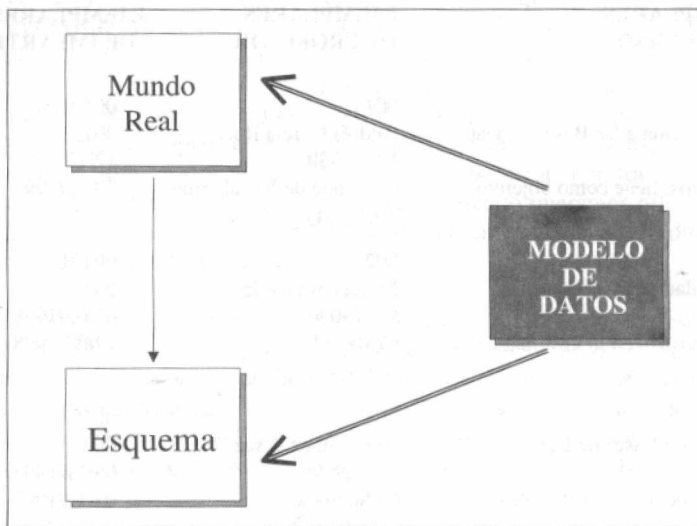


Figura 1.5. Diferencia entre modelo y esquema

⁹ Se propone al lector que, en lo sucesivo, cuando lea, en el campo de la Ingeniería del Software, algún texto relativo a los modelos de datos, lo haga siendo consciente de esta distinción, y podrá comprobar cómo, en la mayor parte de los casos, aparece la ambigüedad que aquí comentamos oscureciendo y dificultando el entendimiento del texto. En la orientación al objeto se llama “modelo de clases”, además de al modelo en sí mismo, a lo que sería, siguiendo las consideraciones aquí expuestas, un esquema de clases.

Es preciso distinguir también entre **esquema**, como descripción de la estructura de la base de datos, y **ejemplar**¹⁰ del esquema, que son los datos que en un determinado momento se encuentran almacenados en el esquema. Así, mediante un modelo de datos podríamos describir los cursos de doctorado que se imparten en nuestro mundo real, obteniendo el esquema que aparece en la figura 1.3 (a), en donde el curso se describe por aquellas características o propiedades que hemos considerado de interés (*Cód_curso*, *Nombre*, *Núm_horas*, ...), lo mismo los profesores, etc.

Los datos concretos en un determinado momento, es decir, el curso con *Nombre* Introducción a las Bases de Datos, cuyo *Cód_curso* es 00101, *Núm_horas* 20, etc. sería un *ejemplar* de CURSO. La colección de ejemplares de todos los elementos de un esquema en un momento determinado constituyen un ejemplar del esquema; en la figura 1.6 hemos representado tres ejemplares de CURSO, al igual que de PROFESOR y de IMPARTE.

EJEMPLARES DE CURSO	EJEMPLARES DE PROFESOR	EJEMPLARES DE IMPARTE
00101	001	00101
Introducción a las Bases de Datos	Andrés García Ruiz	001
030	12312330	12/12/1997
Este curso tiene como objetivo...	C/ Conde de Vistahermosa	20/12/1997
	5.621.333	
00102	002	00101
Seguridad de la información	Mercedes García Arias	003
020	50179030	01/03/1998
La seguridad en la informática...	C/ Río Miño	12/03/1998
	3.928.352	
00203	003	00203
Diseño de Bases de Datos	Julio López Pérez	002
100	52342876	05/11/1997
Dentro de las bases de datos...	C/ Segovia	07/12/1997
	6.564.125	
.....		

Figura 1.6. Ejemplar del esquema descrito en la figura 1.3

¹⁰ Aunque no es el más usado, emplearemos el término **ejemplar** por considerar que es el que mejor expresa la idea relativa a este concepto. “Ocurrencia”, que es el término más extendido, pero no tiene, según el DRAE, un significado coincidente con la idea que tratamos de expresar; también se usa a veces “instancia”, cuyo significado no responde en absoluto a este concepto. *Realización* o *estado* son vocablos mucho más apropiados pero muy poco utilizados.

El esquema es relativamente invariante en el tiempo; mientras no cambie el mundo real (o nuestra interpretación del mismo) el esquema permanece. Sin embargo, los datos, es decir los ejemplares, son distintos en el transcurso del tiempo; por ejemplo, se da de alta un nuevo curso (inserción), desaparece un profesor (borrado), se cambia el profesor que imparte un curso (modificación), etc.

La expresión *base de datos* también se suele utilizar de forma ambigua, ya que con ella se hace referencia unas veces a un determinado ejemplar del esquema, mientras que otras se llama así a todos los valores que puede tomar un esquema en el transcurso del tiempo, es decir, a la serie de ejemplares del esquema. Si analizamos con detenimiento la definición de esquema y de base de datos de DITTRICH (1994) que aparece anteriormente, no queda claro si la base de datos es la colección de datos en un momento determinado (para nosotros, un ejemplar) o es el conjunto de posibles ejemplares, ya que la expresión de Dittrich “la colección de datos en sí misma” puede referirse a cualquiera de las dos cosas. Por ello, consideramos muy acertada la precisión de DATE (1995) que observa que, al igual que en los lenguajes de programación existen *variables* (constituidas por un *tipo* y un *contenido*), las cuales tienen, en un momento determinado, un cierto *valor*, del mismo modo en las bases de datos se debería hablar de *variables de base de datos*, cuyo tipo sería el esquema y su contenido todos los posibles valores del esquema; su valor, en un momento determinado, sería un ejemplar del esquema. Nosotros utilizaremos la expresión “base de datos” en el sentido abstracto de todos los posibles ejemplares (que no se debe confundir con el esquema que es su descripción), y cuando queramos referirnos a su contenido en un cierto momento o bien hablaremos de un ejemplar o bien diremos “la base de datos en el momento i” (BD_i).

La relación entre los conceptos modelo, esquema y ejemplar se representa en la figura 1.7, donde un modelo determinado (entre todos los existentes), como instrumento que ayuda a interpretar la realidad, permite obtener distintos esquemas al aplicarlo a realidades distintas¹¹. Cada uno de estos esquemas será una determinada percepción de una cierta realidad, y podrá tener múltiples ejemplares en el transcurso del tiempo; en un momento determinado, habrá un único ejemplar de dicho esquema.

El esquema, en sí mismo, tiene que estar descrito en términos de datos, por lo que a estos datos se los llama a veces **metadatos**, es decir, datos acerca de los datos. Si los conceptos del modelo de datos son descritos recursivamente utilizando el mismo modelo de datos, el esquema que los describe recibe el nombre de **metaesquema**.

¹¹ Incluso, el mismo modelo aplicado a la misma realidad puede dar lugar a distintos esquemas dependiendo de las distintas interpretaciones que, para el diseñador y/o el usuario, tenga esa realidad.

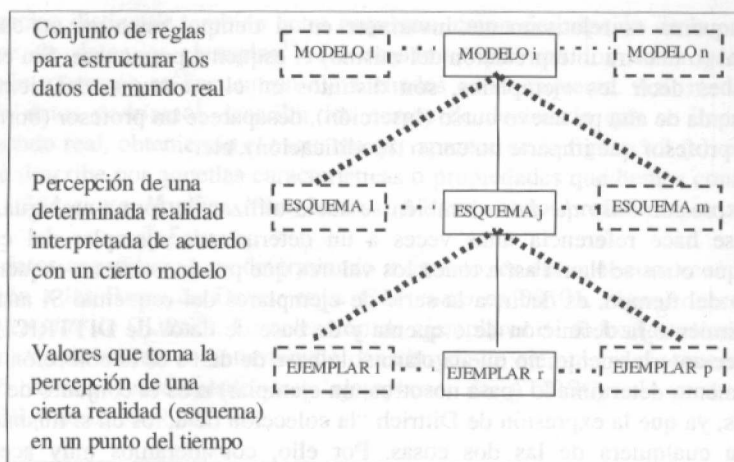


Figura 1.7. Relación entre modelo, esquema y ejemplar

3. TIPOS DE ABSTRACCIÓN EN EL DISEÑO DE BASES DE DATOS

Decíamos anteriormente que el proceso de abstracción nos ayuda a modelar los datos al hacer que nos centremos en lo esencial, pasando por alto aspectos que no consideramos relevantes para nuestros objetivos en la representación del mundo real.

En la figura 1.8 presentamos el concepto de *ambulancia* como una abstracción en la que únicamente recogemos aquellas características (chasis, ruedas, sirena, etc.), comunes a todas las ambulancias y que la distinguen de otros vehículos, que son de interés para nuestros fines.

Los modelos de datos ofrecen distintos tipos de abstracciones a fin de facilitar la representación de los datos; siendo el esquema el resultado de aplicar un proceso de abstracción a un determinado mundo real. Suelen ser cuatro¹² los tipos de abstracción que utilizan los modelos de datos en el diseño de bases de datos: **Clasificación**, **Agregación**, **Generalización**¹³ y **Asociación**¹⁴, las cuales pueden combinarse entre sí ofreciendo interesantes mecanismos semánticos para estructurar los datos.

¹² Algunos autores –ver, por ejemplo ELSMARI (1989)– añaden la identificación; bastantes otros –ver, por ejemplo BORGIDA (1984) y BATINI (1992)– sólo consideran la clasificación, la agregación y la generalización como las abstracciones básicas.

¹³ La generalización no se utiliza en todos los modelos de datos, aunque sí se está introduciendo en extensiones, como ha ocurrido en el modelo E/R y en el estándar SQL: 1999.

¹⁴ La asociación no se utiliza en el modelo relacional, y algunos autores –como TSCHIRITZIS (1982) y BATINI (1992)– no la consideran un tipo de abstracción específico, sino una agregación.



Figura 1.8. Ejemplo de abstracción

Estas abstracciones permiten establecer vinculaciones entre los elementos de un modelo. La primera (clasificación) establece una vinculación entre una categoría de objetos y cada objeto en particular (ejemplar) que pertenece a dicha categoría, mientras que en las otras tres (agregación, generalización y asociación) la relación se establece entre categorías de objetos y, por tanto, también entre los correspondientes ejemplares de dichas categorías.

3.1. Clasificación/Particularización

La clasificación es la acción de abstraer las características comunes a un conjunto de ejemplares para crear una categoría a la cual pertenecen dichos ejemplares.

BRODIE (1984) define la clasificación como “una forma de abstracción en la que una colección de objetos se considera como una clase de objetos de más alto nivel. Una clase de objetos es una caracterización precisa de todas las propiedades compartidas por todos los objetos en la colección. Un objeto es un ejemplar de una clase de objetos si tiene las propiedades definidas en la clase”. Así, podemos abstraer una clase “Mes” definiendo las características comunes a todos los meses (período de tiempo que se mide en días –aproximadamente 30 días– y cuyos límites están bien definidos) y pasando por alto diferencias que no son importantes para nuestros fines (como que unos meses comprenden algunos días más que otros); cada uno de los meses (enero, febrero, etc.) será un ejemplar de la clase “Mes”.

La clasificación se utiliza en el diseño de bases de datos para definir una categoría (*clase* o *tipo*¹⁵) a partir de sus ejemplares, las cuales tienen propiedades comunes por las que se caracterizan. La clase abstraída se puede considerar, de acuerdo con la teoría de conjuntos, la **intensión** (parte definitoria) de todos los posibles ejemplares de dicha clase; la colección de estos ejemplares en un momento determinado constituye una **extensión** de la correspondiente clase. Así, en el ejemplo anteriormente propuesto, describimos a los profesores de nuestra universidad creando una clase (intensión) denominada PROFESOR¹⁶, cuyos ejemplares en un cierto momento (extensión) serán todos y cada uno de los profesores que pertenezcan, en ese momento, a nuestro Universo del Discurso (la gestión de los cursos de doctorado de la universidad).

El proceso inverso de la clasificación es la particularización¹⁷ que consiste en pasar de la clase a sus ejemplares, generando o examinando los distintos objetos particulares a partir de la clase que los describe. Los procesos de clasificación/particularización y un ejemplo de los mismos se representan en la figura 1.9.

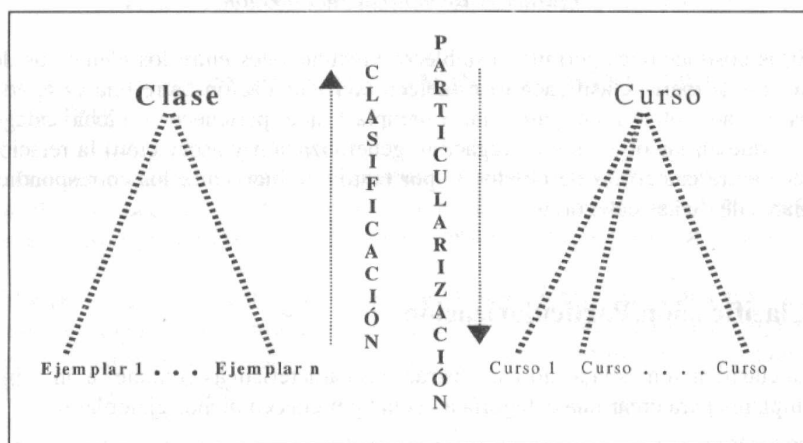


Figura 1.9. Representación de la clasificación/particularización y un ejemplo de la misma

¹⁵ Es más común en los modelos de datos utilizar el término tipo que clase; en los modelos de objetos se suele utilizar clase, aunque en alguno de ellos aparecen clase y tipo como conceptos distintos. Aquí usaremos clase o categoría en un sentido genérico, mientras que en el próximo capítulo, al tratar el modelo Entidad/Interrelación, hablaremos de tipos, que es el término que emplea este modelo.

¹⁶ Utilizaremos las mayúsculas siempre que nos estemos refiriendo a una determinada clase (o tipo) de objetos de nuestro universo del discurso como PROFESOR, CURSO, DEPARTAMENTO, PROGRAMA, etc.

¹⁷ Aunque en bases de datos se utiliza "instanciación" para expresar en castellano esta idea, este término no existe en el DRAE. Además, como hemos dicho anteriormente, tampoco el significado de instancia, según el DRAE, se corresponde en absoluto con el concepto que aquí tratamos de expresar.

La clasificación se corresponde con el concepto de **pertenencia** a un conjunto (*es miembro de*¹⁸).

Los ejemplares de una clase tienen características similares por medio de las cuales describimos la correspondiente clase; estas características toman valores concretos para cada uno de los ejemplares que pertenecen a la clase; por ejemplo, el *Nombre* de un *curso* concreto de la base de datos descrita en la figura 1.3, cuyo ejemplar aparece en la figura 1.6, es *Introducción a las Bases de Datos*, y el *Número_horas* es de 30¹⁹.

Los mismos objetos admiten clasificaciones distintas; así, en el caso de los cursos de doctorado, en lugar de crear una clase CURSO, podríamos haber optado por abstraer dos clases: CURSO_OBLIGATORIO y CURSO_OPCIONAL, o también CURSO_EN_ESPAÑOL y CURSO_EN_INGLES, por ejemplo.

Todos los modelos de datos de las bases de datos admiten la abstracción de clasificación.

3.2. Agregación/Desagregación

La abstracción de agregación consiste en construir un nuevo elemento del modelo como **compuesto** de otros elementos (**componentes**); los componentes “*son parte de*” el elemento compuesto. Así ocurre en la agregación de las clases RUEDA (4), CHASIS (1), SIRENA (1)), ... para obtener la clase AMBULANCIA²⁰ –ver figura 1.10–. Esto supone que también un ejemplar de ambulancia (una ambulancia concreta) es una agregación de ejemplares de las clases compuestas (un determinado chasis, cuatro ruedas y una sirena).

Se pueden considerar tres tipos distintos de agregación:

- **Agregación de clases para obtener una clase compuesta**²¹. Por ejemplo, DEPARTAMENTO se puede modelar, mediante una abstracción de agregación, a partir de la clase ÁREA, considerándolo un conjunto de diferentes áreas, tal como aparece en la figura 1.11; en ella se puede ver también un ejemplar de dicha abstracción, donde el Departamento de Informática es una agregación del área de Lenguajes y Sistemas Informáticos

¹⁸ También se utiliza la expresión “es ejemplar de”.

¹⁹ En los modelos de objetos se permite que existan características que tomen valores para toda la clase, no para cada uno de los objetos en particular; por ejemplo, el número de cursos de doctorado que se imparten en la universidad o la media de horas por curso son características de la clase CURSO, no de sus ejemplares.

²⁰ Se puede llegar a un cierto objeto mediante diferentes procesos de abstracción. Así, anteriormente habíamos modelado la clase AMBULANCIA mediante una *clasificación* a partir de sus ejemplares; ahora abstraemos la clase AMBULANCIA como una agregación de sus componentes.

²¹ El ejemplo de la ambulancia corresponde a este tipo de agregación. La agregación de clases se puede, a su vez, subdividir en varios tipos distintos en los cuales no vamos a entrar en este texto.

(L. y S.I.) y del área de Ciencias de la Computación e Inteligencia Artificial (C.C. e I.A.).

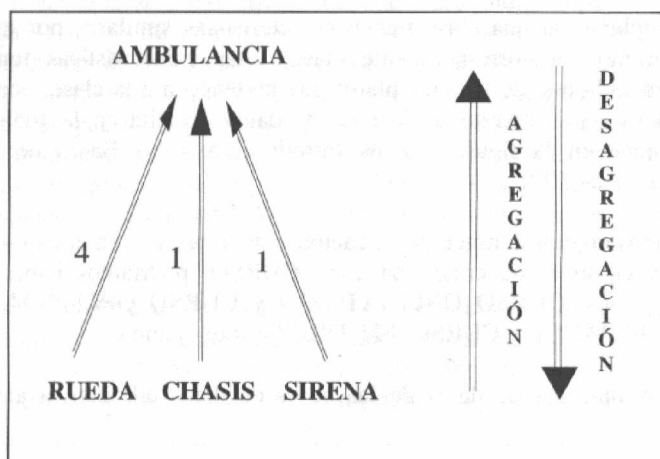


Figura 1.10. Ejemplo de agregación

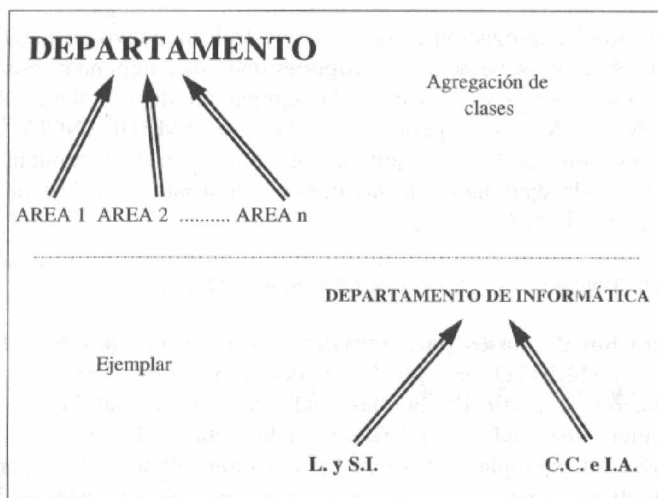


Figura 1.11. Ejemplo de agregación de clases y de un ejemplar de la misma

- **Agregación de propiedades para obtener una clase.** Por ejemplo CURSO se puede considerar como agregación de sus propiedades *Cód_curso*, *Nombre*, *Núm_horas*, etc. –véase figura 1.12–. Esto supone también una agregación de valores de las correspondientes propiedades (un determinado *Nombre*, etc.) para obtener un cierto ejemplar de la clase (un curso en concreto).

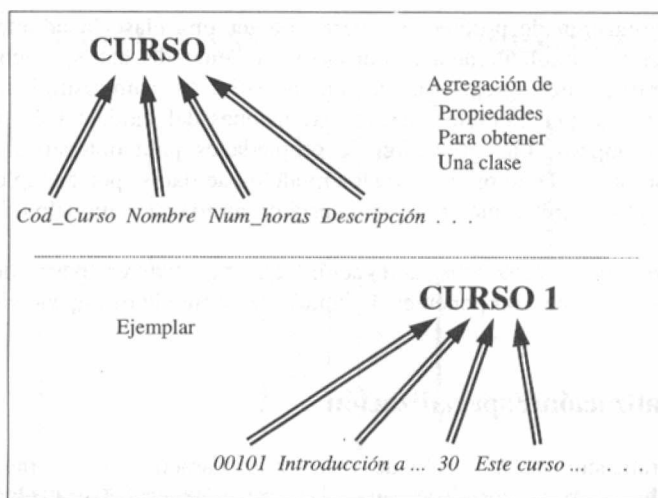


Figura 1.12. Ejemplo de agregación de propiedades y de un ejemplar de la misma

- **Agregación de propiedades para obtener una propiedad compuesta.** Por ejemplo, la agregación de *Día*, *Mes* y *Año* para obtener una *Fecha*. Esto supone una agregación de valores para obtener un valor compuesto –véase figura. 1.13–

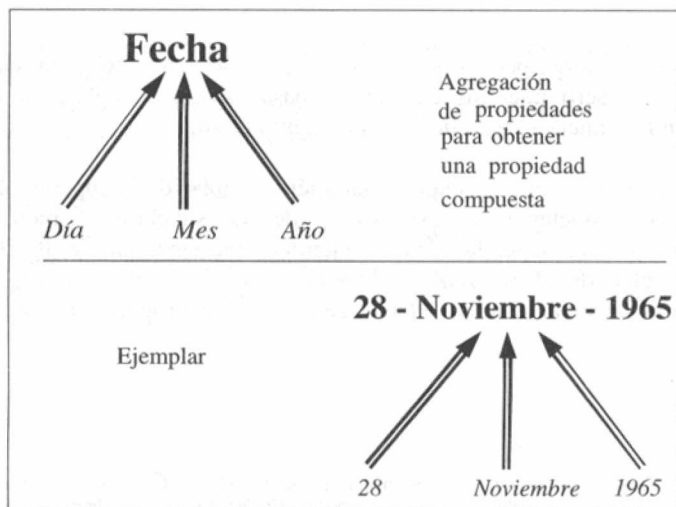


Figura 1.13. Ejemplo de agregación de propiedades para obtener una propiedad compuesta y de la correspondiente agregación en el ejemplar

La agregación de propiedades para obtener una clase la admiten, bien sea explícita o implícitamente, todos los modelos de datos; mientras que el mecanismo de agregación de clases sólo lo suministran los modelos semánticos (por ejemplo, algunas extensiones del modelo E/R) y los modelos de objetos. La agregación de propiedades para obtener una propiedad compuesta no la admiten todos los modelos de datos (por ejemplo, el modelo Codasyl la admite, mientras que no está permitida en el modelo relacional²²).

El proceso inverso a la agregación, que consiste en pasar del elemento compuesto a sus componentes, lo hemos denominado **desagregación**²³.

3.3. Generalización/Especialización

La generalización es la acción de abstraer las características comunes a varias clases (**subclases**) para constituir una clase mas general (**superclase**) que las comprenda a todas; cada generalización es un árbol (jerarquía) de un solo nivel, donde la raíz es la superclase y las hojas son las subclases; el proceso inverso de la generalización es la especialización, en la que se pasa de la superclase a las subclases. Por ejemplo, se puede generalizar las clases PROFESOR y ESTUDIANTE en la superclase PERSONA, la cual tendrá las características comunes a ambas subclases, como el *Código*, *Nombre*, *Apellidos*, etc.; también podríamos, a partir de PERSONA, pasar a las clases PROFESOR y ESTUDIANTE mediante una especialización –véase figura 1.14–.

La generalización/especialización es un proceso parecido a la clasificación/particularización, pero mientras en ésta se pasa de los ejemplares a la clase (o viceversa), en la primera se pasa de una clase a otra clase.

Todo ejemplar de una subclase es también ejemplar de la superclase y, además de poseer las características específicas de la subclase, hereda todas las correspondientes a la superclase²⁴; por ejemplo, un ejemplar de PROFESOR es también ejemplar de PERSONA y hereda el *Código*, el *Nombre*, etc. de la correspondiente persona, teniendo además características propias como *Materia*, *Tipo*, etc.

²² El modelo relacional, tal como fue inicialmente propuesto –CODD (1970)– no admite los agregados de datos –*atributos compuestos*–; en cambio, la versión 2 del modelo, CODD (1990), sí los admite, lo que es criticado por DATE (1992).

²³ Aunque en los modelos de objetos se suele utilizar el término *desensamblaje*, hemos considerado preferible, al menos cuando estamos presentando el concepto de modelo de datos en general y no un modelo concreto, denominar *desagregación* al proceso inverso a la agregación.

²⁴ En algunos modelos de objetos se permite la *inhibición de herencia* (mediante la cual se puede no heredar **todas** las características de la superclase).

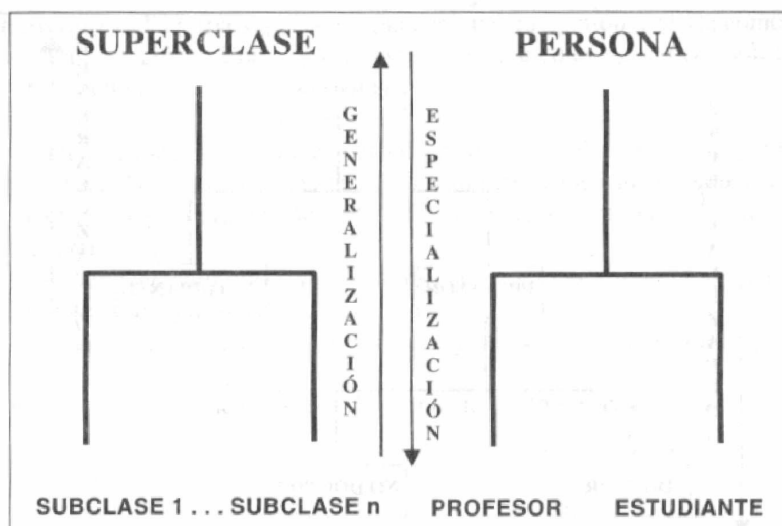


Figura 1.14. Representación y ejemplo de generalización/especialización

El conjunto de ejemplares de una subclase “es un” subconjunto de los ejemplares de la correspondiente superclase, por lo que se crea la expresión “ES_UN” (en inglés “IS_A”) para denominar estas jerarquías que tuvieron su origen en el campo de la inteligencia artificial.

Se admite la aplicación de sucesivas generalizaciones/especializaciones formando jerarquías de varios niveles tal como aparece en la figura 1.15, donde se combina la generalización de ESTUDIANTE y PROFESOR en PERSONA y la especialización de PROFESOR en DOCTOR y NO_DOCTOR.

Aunque esta abstracción es muy intuitiva y muy útil en el proceso de diseño por la semántica que permite captar, se ignora en bastantes modelos de datos; aquellos que la admiten suministran distintos mecanismos de generalización a los que se pueden aplicar distintos tipos de restricciones²⁵.

²⁵ En el siguiente capítulo, al estudiar el modelo E/R, volveremos de nuevo a esta abstracción, analizando con más profundidad sus características.

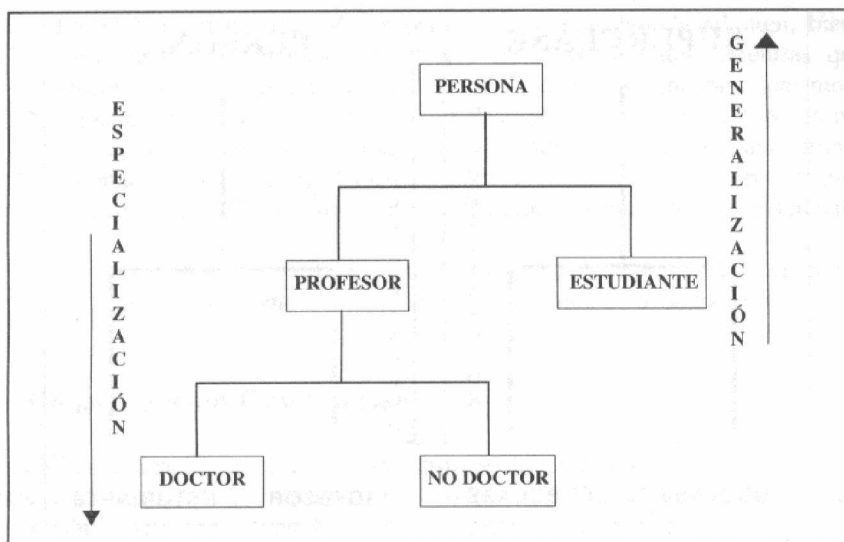


Figura 1.15. Ejemplo de jerarquía a dos niveles con una generalización y una especialización

3.4. Asociación/Disociación

Es una abstracción que se utiliza para vincular dos o más clases (y, por tanto sus ejemplares), creándose un elemento de un tipo distinto. Por ejemplo, en la figura 1.16 aparece la asociación **Imparte** entre las clases PROFESOR y CURSO (un determinado profesor “imparte” un cierto curso y un determinado curso “se imparte” por un cierto profesor); “imparte” es un elemento de un tipo distinto de PROFESOR y CURSO, y tampoco, en nuestra opinión, es una agregación de las mismas como consideran algunos autores.

Existen notables diferencias entre la asociación y las abstracciones anteriormente estudiadas; diferencias que se escapan al alcance de este libro, pero que llevan a que, como ya hemos señalado, no siempre se incluya la asociación entre las abstracciones.

Ciertos autores como BORGIDA (1984), no consideran la asociación entre los tipos de abstracción, mientras que para otros, como TSICHRITZIS (1982), la asociación es una agregación. En algunos modelos de datos tampoco aparece esta abstracción como tal, no existiendo ningún concepto especial para representarla²⁶. Para nosotros, aun cuando consideramos que la asociación puede parecerse a la agregación,

²⁶ Por ejemplo, en el modelo relacional, tanto las clases como las asociaciones entre clases se representan por medio de **relaciones**, no haciéndose distinción alguna entre unas y otras.

tiene ciertos rasgos distintivos que aconsejan, en nuestra opinión, tratarla como un tipo diferente de abstracción²⁷. Entre otras, existen las siguientes diferencias entre ambos tipos de abstracción:

- Cuando se asocian dos o más categorías, el nuevo elemento que aparece tiene determinadas características que lo distinguen de las categorías normales, por lo que, en general, los modelos de datos crean un nuevo concepto para representarlo.
- El nuevo elemento no *está compuesto*, como en el caso de la agregación, por los elementos que asocia.
- En la agregación puede existir herencia, y no así en la asociación.

Al proceso inverso a la asociación lo hemos llamado *disociación*.

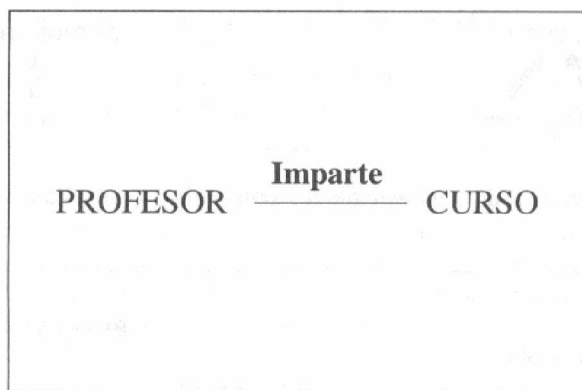


Figura 1.16. Ejemplo de asociación

3.5. Jerarquías de abstracciones

En el proceso de modelado de una determinada realidad, es preciso a menudo combinar distintas abstracciones, formando lo que se conoce como una jerarquía de abstracciones. En la figura 1.17 se combina la agregación de propiedades con la generalización, apareciendo la clase PERSONA como una **agregación** de sus características (*DNI*, *Nombre*, *Dirección*) y también como una generalización de PROFESOR y ESTUDIANTE, éstos además de las características heredadas (*DNI*, *Nombre*, *Dirección*), pueden tener características propias (PROFESOR tiene *Materia* y *Tipo*; ESTUDIANTE tiene *Curso*). Vemos, por consiguiente, que una misma clase

²⁷ Un análisis más en profundidad de este tipo de abstracción se hará en el próximo capítulo, al estudiar el modelo Entidad/Interrelación, ya que la asociación es un concepto fundamental en este modelo, en el que la nueva clase creada (así como los ejemplares de la misma) constituyen elementos distintos del modelo que reciben el nombre de **interrelación**.

(PERSONA) se puede abstraer tanto por agregación de propiedades como por generalización de otras clases.

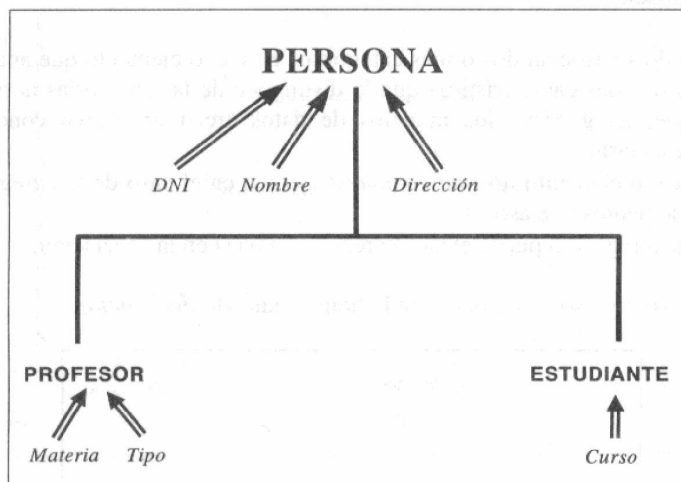


Figura 1.17. Combinación de agregación y generalización

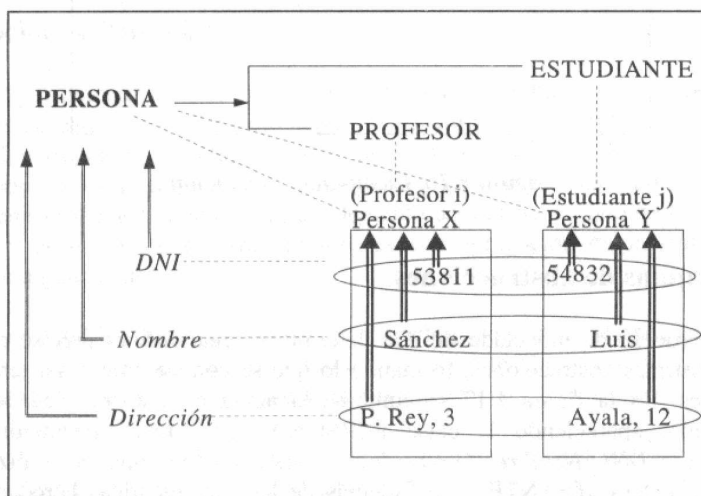


Figura 1.18. Ejemplo de abstracciones de clasificación, agregación y generalización

También es posible abstraer una categoría por clasificación de sus ejemplares. En la figura 1.18 la categoría PERSONA se obtiene, además de por generalización o por agregación como en el ejemplo anterior, por clasificación de sus ejemplares (persona

X, Y, ...); cada una de las subclases PROFESOR y ESTUDIANTE se pueden obtener también por clasificación de sus respectivos ejemplares.

4. CONCEPTO DE MODELO DE DATOS

Aunque existen muchos modelos de datos es posible abstraer una serie de características comunes a todos ellos, definiendo así el concepto de modelo de datos en general, que posteriormente se ha de particularizar para describir cada modelo en concreto.

Podemos ya definir de forma más precisa el concepto de modelo de datos como *“un conjunto de conceptos, reglas y convenciones bien definidos²⁸ que nos permiten aplicar una serie de abstracciones a fin de describir y manipular²⁹ los datos de un cierto mundo real que deseamos almacenar en la base de datos”*.

Los modelos de datos facilitan la creación de categorías mediante la aplicación de los tipos de abstracción anteriormente considerados, lo que lleva a diferenciar dos tipos de modelos:

- *Modelos de datos estrictamente tipados*, en los que cada dato tiene que pertenecer forzosamente a una categoría previamente definida en el esquema; los datos (ejemplares) que no pertenecen a una categoría no pueden ser manejados por el modelo. En algunos casos, no se permite la evolución de las categorías, y los datos tienen que permanecer en la categoría en la que fueron creados.
- *Modelos de datos débilmente tipados*, en los que no es obligatorio que los datos (ejemplares) pertenezcan a categorías, sino que pueden existir por sí mismos. Se permite la existencia de categorías en aquellos casos en que es conveniente para representar ciertas extensiones. Las categorías, si existen, se tratan como los datos individuales.

En las bases de datos se usan modelos estrictamente tipados, dado que, a pesar de sus inconvenientes, en especial su falta de flexibilidad, tienen como ventaja el posibilitar el tratamiento de grandes cantidades de datos al agruparlos en categorías. Éstos son los modelos de datos a los que nos referiremos en este libro.

²⁸ A veces se añade “matemáticamente”, es decir se considera que los conceptos de un modelo de datos han de estar “bien definidos matemáticamente”, en la realidad muchos modelos de datos han surgido en la práctica soportados por los correspondientes Sistemas de Gestión de Bases de Datos (SGBD), no habiendo sido por tanto formalmente definidos; por esta razón, aunque otros modelos de datos, como el relacional, sí han sido definidos en términos matemáticos, nosotros hemos preferido no introducir esta restricción que no cumplen todos los modelos.

²⁹ En algunos casos no se incluye la manipulación como parte del modelo, sino que sólo se consideran sus aspectos estáticos, especialmente en aquellos orientados al diseño de alto nivel (modelos conceptuales).

Un modelo de datos define las reglas según las cuales han de ser estructurados los datos acerca del mundo real. Como ya hemos dicho, la representación de una determinada realidad mediante un modelo (instrumento que nos facilita el proceso de representación) da lugar a un esquema, el cual describe las categorías existentes en dicha realidad. Sin embargo, la realidad no contempla sólo aspectos estáticos, como son aquellos que se representan en el esquema, sino también propiedades dinámicas, ya que los ejemplares de las categorías varían en el transcurso del tiempo, y estas propiedades dinámicas han de ser también especificadas en operaciones de consulta y actualización de la base de datos.

Por tanto, podemos decir que las propiedades del mundo real son de dos tipos:

- *Estáticas*, o relativamente invariantes en el tiempo, que responden a lo que se suele entender como estructuras.
- *Dinámicas*, que son las operaciones que se aplican a los datos o valores almacenados en las estructuras, los cuales varían en el transcurso del tiempo al aplicárseles dichas operaciones.

El modelo de datos ha de proporcionar facilidades para recoger ambos aspectos, por lo que se define formalmente como el par:

$$MD = \langle G, O \rangle$$

donde **G** es el conjunto de reglas de generación que permiten representar la componente estática, es decir, describir las estructuras de nuestro universo del discurso, y **O** es el conjunto de operaciones autorizadas sobre la correspondiente estructura, operaciones que permiten representar la componente dinámica.

La componente estática de un determinado modelo de datos expresado con una sintaxis es el *Lenguaje de Definición de Datos (LDD)*, y la componente dinámica el *Lenguaje de Manipulación de Datos (LMD)*; ambos constituyen el *Lenguaje de Datos (LD)*³⁰.

Un modelo de datos define reglas generales para especificar las estructuras de datos y las operaciones permitidas sobre los datos³¹; estas operaciones han de ser ejecutadas en el contexto proporcionado por las estructuras.

Analicemos a continuación, con mayor detalle, cada uno de estos componentes.

³⁰ Los SGBD suelen tener además un Lenguaje de Consulta (en inglés "Query Language" –QL–) y un *Lenguaje de Control* (en inglés Control Language).

³¹ En los modelos conceptuales la componente dinámica o bien no se define o se le concede poca importancia; en cambio, en los modelos de implementación (modelos convencionales), la componente dinámica es tan importante como la estática.

4.1. Estática

La estática de un modelo de datos está compuesta por:

- A) **Elementos permitidos.** No son los mismos para todos los modelos de datos (varían especialmente en terminología), pero en general son:

- A1) **Objetos**³² (entidades, relaciones, registros, etc.)
- A2) **Asociaciones** entre objetos (interrelaciones, “set”, etc.)
- A3) **Propiedades** o características de los objetos o asociaciones (atributos, campos, elementos de datos, etc.)
- A4) **Dominios**, que son conjuntos nominados de valores homogéneos sobre los que se definen las propiedades.

A estos elementos permitidos se les podrán aplicar aquellas abstracciones reconocidas por el modelo. La representación de estos elementos depende de cada modelo de datos, pudiendo hacerse en forma de **grafos** (como en el modelo E/R, o en el Codasyl) o de **tablas** (como en el modelo relacional).

- B) **Elementos no permitidos o restricciones.** No todos los valores, cambio de valor o estructuras están permitidos en el mundo real; por ejemplo, un niño de tres años no puede estar casado, ni una persona puede pasar directamente de soltera a viuda, etc. Además, cada modelo de datos también impone por sí mismo limitaciones a las estructuras que admite³³. Estas limitaciones, que unas veces vienen impuestas por el mismo modelo de datos y otras nos las impone el universo del discurso que estamos modelando, se denominan *restricciones*; las que son impuestas por el modelo son *restricciones inherentes* y las que responden al deseo de que el sistema de información sea un reflejo lo más fiel posible del mundo real son las *restricciones de integridad o semánticas*. Existen, por tanto, dos tipos de restricciones:

- B1) **Restricciones inherentes** (del modelo), son aquellas que vienen impuestas por la misma naturaleza del modelo de datos, el cual no admite ciertas estructuras, introduciendo así rigideces a la hora de modelar. El usuario (diseñador) no define estas restricciones, siendo el SGBD, en el que subyace el modelo, el que impide, en el momento de la definición del esquema, que se introduzcan estructuras no admitidas por el correspondiente modelo.

³² En este capítulo, que trata de los modelos de datos en general, hemos preferido referirnos a objetos en lugar de a entidades, término que se suele asociar a un determinado modelo (el Modelo Entidad/Interrelación); tampoco hemos querido emplear el vocablo “cosas”, como hacen algunos autores, por ejemplo KENT (1978). No entramos aquí en las dificultades que entraña dar una definición rigurosa de lo que es un objeto dentro de este contexto, ni en la diferenciación o similitud de objeto con asociación o propiedad; por ahora, para nosotros, un objeto tiene, como ya hemos dicho, el significado, que conoce cualquier lector, del lenguaje común.

³³ El modelo relacional, por ejemplo, no permite que dos ejemplares (filas) de una tabla sean iguales.

B2) Restricciones de integridad o semánticas³⁴ (de usuario), son aquellas que permiten captar la semántica del universo del discurso que se quiere modelar y verificar la corrección de los datos almacenados en la base. El usuario (diseñador) ha de definir, y a veces programar, estas restricciones a fin de rechazar ciertas asociaciones o de limitar los valores que pueden tomar los datos de la base de datos o de impedir ciertos cambios de los mismos. Según los instrumentos que proporcione el modelo de datos para definir y gestionar las restricciones, éstas pueden ser:

- **Reconocidas por el MD.** Su definición le corresponde al diseñador, pero su gestión es responsabilidad del modelo de datos, el cual las reconoce y recoge en el esquema, suministrando instrumentos para su definición y obligando a su cumplimiento.
- **Ajenas al MD.** Son, por completo, responsabilidad del diseñador, ya que el modelo de datos no las reconoce ni proporciona instrumentos para manejarlas.

Podemos, definir la componente estática del modelo de datos como el par:

$$G = \langle G_e, G_r \rangle$$

donde G_e es el conjunto de reglas de generación de estructuras (objetos del modelo y restricciones inherentes) y G_r es el conjunto de restricciones de usuario.

La aplicación de la componente estática (reglas de generación) de un modelo de datos a un determinado Universo del Discurso (UD) nos da como resultado un esquema, que es la estructura de datos que describe, en el correspondiente modelo, las categorías que han resultado de las abstracciones aplicadas al mundo real que se trata de modelar; es decir:

$$G[UD] = E$$

4.2. Dinámica

El conjunto de valores que toman las distintas categorías de un esquema en un momento determinado t_i recibe el nombre de ejemplar del esquema o estado de la base de datos³⁵ en el tiempo t_i (BD_i); en otro momento t_j el ejemplar del esquema será BD_j . Si entre t_i y t_j se ha producido un cambio en algún valor de la base de datos (alta, baja

³⁴ En el siguiente epígrafe hacemos un estudio más completo de estas restricciones.

³⁵ Ya hemos advertido que también se denomina instancia y, a veces, se habla simplemente de base de datos en el momento t_i .

o modificación) $BD_i \neq BD_j$. Tanto BD_i como BD_j deben ser ejemplares válidos de la base de datos, es decir, los valores de ambos deben pertenecer a alguna de las categorías definidas en el esquema³⁶ y cumplir las restricciones de integridad (también deben cumplir, en caso de que existan, las posibles restricciones asociadas al cambio de estado).

La componente dinámica del modelo consta de un conjunto de operadores que se definen sobre la estructura del correspondiente modelo de datos, ya que no todas las estructuras admiten el mismo tipo de operaciones. La aplicación de un operador a un ejemplar de un esquema transforma éste en otro ejemplar:

$$O [BD_i] = BD_j$$

Pudiendo ser $BD_i = BD_j$, por ejemplo en caso de consulta o cuando falla una operación por haberse producido un error³⁷.

Una operación tiene dos componentes:

1. **Localización** o “enfoque”, consiste en localizar un ejemplar de un objeto indicando un camino, o un conjunto de ejemplares especificando una condición. En el primer caso se trata de un sistema **navegacional**, mientras que el segundo se dice que es de **especificación**.
2. **Acción**, que se realiza sobre el(los) ejemplar(es) previamente localizado(s) mediante una operación de localización, y puede consistir en una recuperación o en una actualización (inserción, borrado o modificación).

Sin seguir una sintaxis concreta, sino más bien en un plano conceptual, podemos expresar una sentencia del LMD de la siguiente forma:

LOCALIZACIÓN <condición>

ACCIÓN <objetivo>

donde LOCALIZACIÓN y ACCIÓN³⁸ son mandatos del LMD, <condición> representa una expresión lógica que deben cumplir los objetos que se desea localizar o

³⁶ Ya hemos dicho que nos estamos refiriendo a modelos de datos estrictamente tipados.

³⁷ Si consideramos que el estado de la base de datos viene determinado no sólo por los valores que toman los objetos del esquema, sino también por los valores de sus indicadores (por ejemplo el indicador de error), cualquier operación hace variar el estado de la BD, bien porque cambian los valores de los objetos (en caso de una actualización), bien porque cambian los indicadores (en caso de fallo o de consulta). En algunos MD, como el Codasyl, la manipulación de los datos está basada en los indicadores.

³⁸ La distinción entre localización y acción es de tipo formal; y si bien algunos lenguajes, como el LMD de Codasyl, tienen dos mandatos distintos para expresar la selección y la acción, distinguiendo claramente entre ambos tipos de operación, otros lenguajes, como el SQL, reúnen ambas operaciones en un único operador.

señala el camino que permite llegar a esos objetos, mientras que <objetivo> indica los objetos (o las propiedades de éstos) sobre los que se aplica la acción³⁹.

5. RESTRICCIONES DE INTEGRIDAD

En el mundo real existen ciertas *reglas* que deben cumplir los elementos en él existentes; por ejemplo, una persona sólo puede tener un número de DNI y una única dirección oficial (la que figura en el padrón); además, un número de DNI sólo puede corresponder a una única persona, etc. Cuando diseñamos una base de datos deseamos que ésta refleje lo mas fielmente posible el universo del discurso que estamos tratando de recoger en nuestro sistema de información, por lo que en el esquema de la base de datos, junto con los objetos, las asociaciones y las propiedades de los mismos, debemos describir también estas reglas, llamadas restricciones semánticas o de integridad⁴⁰, las cuales pueden ser definidas como condiciones que limitan el conjunto de ejemplares válidos de un esquema.

La *semántica* y la *integridad* son conceptos que en las bases de datos suelen ir asociados, aunque no son idénticos. Con el término semántica nos referimos al significado de los datos y con el de integridad a la corrección de los mismos y a su consistencia respecto al mundo real del cual proceden. Cuando en el esquema de una base de datos se encuentra descrita la semántica del mundo real, será posible comprobar si los valores de los datos se atienen o no a esta semántica previamente definida, comprobándose la integridad de los mismos; de ahí que digamos que ambos conceptos suelen ir unidos. Nosotros utilizaremos indistintamente las expresiones *restricciones semánticas* o *restricciones de integridad* o, a veces, diremos simplemente *restricciones* cuando por el contexto se comprenda que no nos estamos refiriendo a las restricciones inherentes.

La semántica de los datos, es decir todo lo que conocemos acerca de los datos, se encontraba en un principio en la mente del usuario, el cual comprobaba manualmente si los datos cumplían o no las reglas a ellos asociadas; después fue migrando desde la mente del usuario a los programas; y, por último, ha pasado desde éstos a la base de datos, tal como se muestra en la figura 1.19.

³⁹ Si el SGBD se adaptase estrictamente a la arquitectura a tres niveles de ANSI, el <objetivo> sería el nombre de un esquema externo previamente definido; sin embargo, algunos SGBD, especialmente los basados en el modelo relacional, no obligan a definir previamente el esquema externo, permitiendo describir el objetivo dentro de la misma sentencia de manipulación.

⁴⁰ Algunos autores –véase, por ejemplo, DATE (1995)– prefieren llamarlas reglas, reservando el término “restricción” para la condición que se debe satisfacer; sin embargo, la denominación más habitual es la de restricción.

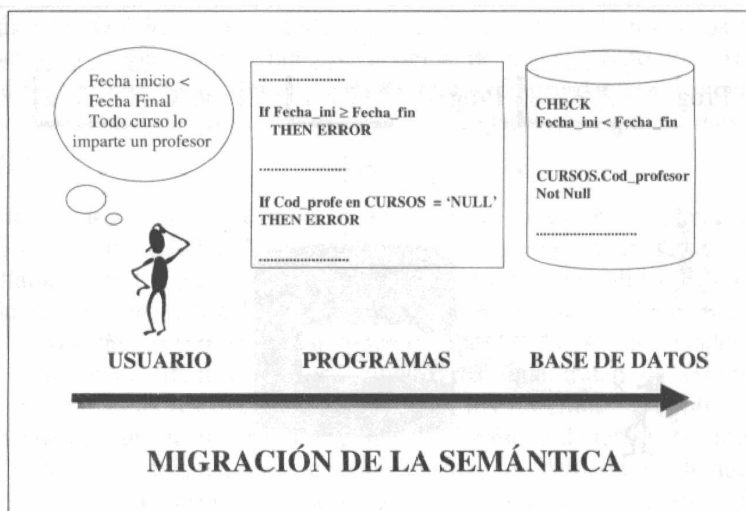


Figura 1.19. Migración de la semántica de los datos

Son muchas las ventajas de tener integrada la descripción de las restricciones junto con la de los datos en el esquema de la base de datos, en lugar de que esté dispersa entre los diferentes programas de aplicación. Por un lado, tiene ventajas relativas a la integridad, ya que al ser única la descripción de las restricciones no se pueden producir inconsistencias debidas a que los distintos programadores hayan definido, cada uno en su programa y no de manera uniforme, las restricciones de integridad (por ejemplo, un programador se puede olvidar de incluir en su aplicación una determinada comprobación que otros sí han incluido); de esta forma puede, además, disminuirse drásticamente la carga de programación (se considera que la programación de las sentencias necesarias para controlar la integridad puede llegar a suponer en algunos casos hasta un 90% del total de una aplicación).

Por otro lado, tiene también ventajas semánticas, ya que es importante que el significado de los datos, como parte fundamental de los mismos, se encuentre descrito junto con el resto de sus características y que sea únicamente el diseñador el que se ocupe, por una sola vez, de definir la semántica, no dejando esta responsabilidad en manos de los programadores de aplicaciones; lo cual evita redundancias y permite a los usuarios, siempre que tengan la debida autorización, conocer el significado de los datos sólo con consultar el esquema de la base de datos en lugar de tener que indagar en las diferentes aplicaciones, tal como se muestra en la figura 1.20.

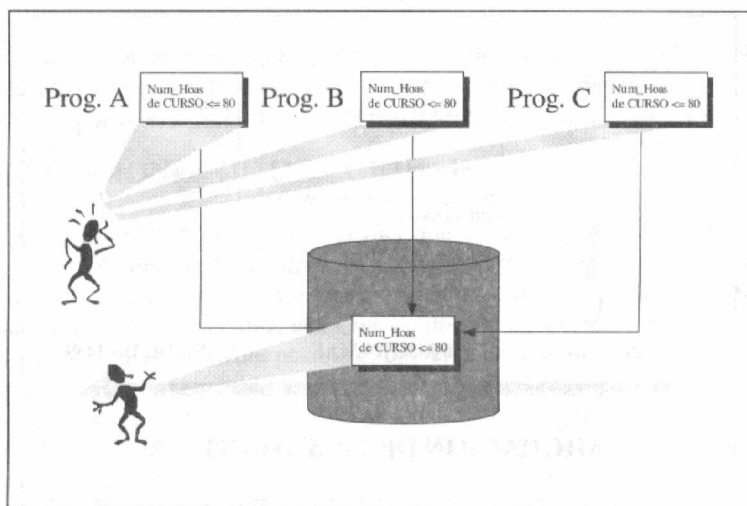


Figura 1.20. Semántica de los datos “dispersa” entre los diferentes programas de aplicación, frente a semántica integrada en la base de datos

Todas estas razones nos muestran la necesidad de que la semántica del mundo real se encuentre descrita en el esquema, es decir, esté centralizada en lugar de dispersa en los diferentes programas de aplicación que actualizan la base de datos; pero, para conseguir este objetivo, los modelos de datos han de permitir representar las restricciones semánticas dando instrumentos para ello, y los SGBD en los que están soportados los modelos tienen que reconocer y gestionar estas restricciones. Las restricciones semánticas que pueden ser especificadas en un determinado modelo de datos y representadas en sus esquemas decimos que son restricciones semánticas *propias*⁴¹ del modelo.

Sin embargo, ningún modelo de datos es capaz de capturar toda la semántica del mundo real mediante restricciones propias, por lo que, en general, es necesario tener restricciones adicionales que no estarán soportadas por el modelo de datos, a las que llamamos restricciones semánticas *ajenas* al correspondiente modelo. También debemos precisar que ciertas restricciones propias de un modelo no son a veces soportadas por algunos SGBD basados en ese modelo⁴².

Tampoco debemos confundir, en especial en el caso del relacional, el modelo con el estándar de un cierto lenguaje para ese modelo, como es el SQL; por lo general, serán distintas las restricciones propias de un estándar y las de los diferentes productos comerciales basados en él.

⁴¹ Que no hay que confundir con las restricciones “inherentes”.

⁴² Por ejemplo, la semántica de los dominios no está totalmente soportada en estos momentos en casi ningún SGBD comercial, a pesar de que los dominios siempre han sido elementos propios del modelo relacional.

Según CODD (1979), la tarea de capturar la semántica de los datos no termina nunca; por esta razón, surgen nuevos modelos de datos (como los orientados al objeto), al tiempo que los modelos existentes (como ocurre con el relacional y con el E/R) se van extendiendo con el objetivo de ser capaces de capturar más elementos semánticos.

Las restricciones ajenas al modelo no son otra cosa que programas o procedimientos escritos en cualquier lenguaje de propósito general (lenguaje anfitrión) con posibles llamadas al lenguaje de manipulación de datos. Su finalidad es comprobar la corrección de una operación de actualización, impidiendo que la violación de una cierta regla existente en el mundo real dé lugar a que los datos almacenados en la base de datos sean inconsistentes con ese mundo real que tratan de representar; sin embargo, al estar estas restricciones dispersas en los programas de aplicación tienen los inconvenientes que acabamos de exponer. La finalidad de las restricciones propias de un modelo es la misma, pero en este caso el modelo da facilidades para su definición y considera esta definición como parte integrante de su esquema. Un lenguaje estándar que asuma completamente un modelo debe proporcionar facilidades (es decir, una sintaxis) para definir todos los elementos del modelo (por tanto, todas las restricciones reconocidas por éste). Del mismo modo, un producto basado en un modelo o en un estándar debe ser capaz de reconocer y gestionar todas las restricciones propias del modelo o del estándar. En general, cuando en el campo de las bases de datos se habla de restricciones de integridad se está haciendo referencia a las restricciones de integridad propias del modelo; además, es habitual suponer que el correspondiente SGBD asume totalmente el modelo, aunque ello no sea cierto en muchos casos.

5.1. Componentes de una restricción

En general, en una restricción de integridad es posible distinguir los siguientes componentes:

- La *operación* de actualización (inserción, borrado o modificación) cuya ejecución ha de dar lugar a la comprobación del cumplimiento de la restricción.
- La *condición*⁴³ que debe cumplirse, la cual es en general una proposición lógica, definida sobre uno o varios elementos del esquema, que puede tomar uno de los valores de verdad (cierto o falso⁴⁴).

⁴³ Es lo que hemos dicho que, a veces, se llama propiamente restricción.

⁴⁴ Podría también tomar el valor de verdad “quizás” en el caso de admitirse valores nulos, teniendo entonces que evaluarse la condición mediante una lógica trivaluada. Obviamos aquí este tema de los valores nulos o información “faltante”, porque introduce una importante complejidad adicional sin aportar nada al concepto de las restricciones de integridad en sí mismo.

- La *acción* que debe llevarse a cabo dependiendo del resultado de evaluar la condición⁴⁵.

Las restricciones de integridad se pueden considerar, en cierto modo como reglas ECA (Evento, Condición, Acción), en las cuales, al ocurrir un evento (en este caso una actualización), se comprueba una condición y dependiendo de su resultado se pone en marcha una acción (rechazar la operación, informar al usuario, corregir el error, etc.)⁴⁶.

Además de estos elementos, las reglas de integridad pueden tener un nombre por medio del cual es posible identificarlas, y también puede indicarse el *momento* en el que ha de evaluarse la condición (antes o después de la operación, de forma inmediata o diferida al final de una transacción, etc.).

En bastantes casos es posible prescindir de alguno de estos componentes, de modo que, por defecto, se tome una cierta opción, simplificando así la definición de las restricciones.

Las restricciones han de ser definidas en la fase de diseño y el cumplimiento de la condición tiene que ser verificado en la de ejecución cuando se está procesando la operación de actualización que provoca cambios en el estado de la base de datos. En una restricción es por tanto necesario distinguir lo que ocurre en la fase de definición y en la de ejecución (actualización de la base de datos).

- *Fase de definición.* En ella, el diseñador ha de describir la restricción especificando sus componentes. El sistema debe comprobar que la definición de la restricción es correcta (respecto al modelo de datos soportado por el sistema) y que el conjunto de restricciones es consistente en sí mismo; por ejemplo, el diseñador puede, por equivocación, definir una restricción sobre atributos inexistentes o imponer una restricción prohibiendo valores nulos para un cierto atributo y en otra restricción declarar una acción de puesta a nulos para ese mismo atributo; en estos casos, el sistema debería detectar la falta de corrección o de consistencia de las restricciones. Una vez comprobada la validez de una restricción, ésta debe ser compilada, junto con los otros elementos, por el SGBD e incluida en el esquema.
- *Fase de ejecución.* En el momento de la ejecución de una sentencia de actualización sobre la que se ha definido una restricción en la que están implicados elementos que van a ser actualizados, es preciso que el sistema

⁴⁵ En general, la acción se desencadena en caso de fallo de la condición, es decir, la respuesta se produce ante un intento de violación de la restricción, aunque –como veremos posteriormente– en ciertos tipos de restricciones como son los *disparadores* la acción se desencadena cuando se cumple la condición.

⁴⁶ Son los mismos componentes de las reglas en las bases de datos activas, pero es que la integridad no es otra cosa que un caso especial de actividad, en la que la operación desencadenante es siempre una actualización, mientras que, en las bases de datos activas, la operación desencadenante puede ser de otro tipo.

compruebe la condición a fin de que si se estuviese produciendo un intento de violación poner en marcha la acción especificada en el momento indicado.

DATE (1990) hace un estudio de la integridad de las bases de datos, proponiendo un enfoque hacia un lenguaje general para formular reglas de integridad en forma declarativa.

5.2. Clasificación de las restricciones

Es muy difícil hacer una clasificación rigurosa de las restricciones, ya que éstas varían mucho dependiendo de los modelos y de los productos; tampoco en los trabajos consultados hemos encontrado una clasificación completa y consistente de las mismas; en la figura 1.21 proponemos una jerarquía de clasificación de las restricciones⁴⁷.

Dentro de las restricciones semánticas, las que hemos llamado *ajenas al modelo de datos*, son procedimientos específicos incluidos en los programas de aplicación a fin de recoger la semántica del UD, que permiten comprobar la consistencia de los datos de la base (obsérvese que su creación está ligada a la función de manipulación y no a la de definición). No están almacenados en el esquema de la BD y, por tanto, pueden ser violadas en actualizaciones en las que no se haya programado la correspondiente restricción. Por esta razón el SGBD tampoco tiene conocimiento de su existencia y el optimizador no puede tomarlas en consideración⁴⁸. Suponen una importante carga de programación y de mantenimiento —ya que en general son procedimentales—, a cambio de lo cual proporcionan el máximo de flexibilidad. Como su nombre indica, son totalmente ajenas al modelo de datos; pero los productos pueden dar facilidades para definirlos. Dentro de ellas se puede distinguir las que se embeben en programas de propósito general y las que constituyen facilidades proporcionadas por algún módulo o lenguaje del SGBD⁴⁹ (no por el núcleo, ya que en este caso no se trataría de restricciones ajenas al modelo).

⁴⁷ La caracterización de las restricciones inherentes ya se ha expuesto cuando se ha definido la estática de un modelo de datos, por lo que en este epígrafe nos limitaremos a estudiar las restricciones semánticas; aunque en la figura 1.21 y en el cuadro que aparece al final del epígrafe también se han incluido las restricciones inherentes.

⁴⁸ El optimizador de los SGBD relacionales tiene como objetivo la búsqueda, para cada sentencia de manipulación, de técnicas eficientes de acceso a los datos almacenados. En los lenguajes navegacionales el usuario indica el camino a seguir para acceder a los datos, pero en los lenguajes de especificación es un componente del SGBD, el optimizador (o planificador de consultas), el que ha de ocuparse de esta tarea. Si el modelo de datos conoce una determinada restricción, el optimizador podrá apoyarse en ella a fin de mejorar la eficiencia en el acceso a la base de datos; si las restricciones, aunque existan, son ajenas al SGBD, éste no puede revisar todos los programas para tener conocimiento de estas restricciones y, por tanto, el optimizador no las tendrá en cuenta.

⁴⁹ Por ejemplo, el SQL Forms de Oracle da facilidades para la definición de la integridad referencial.

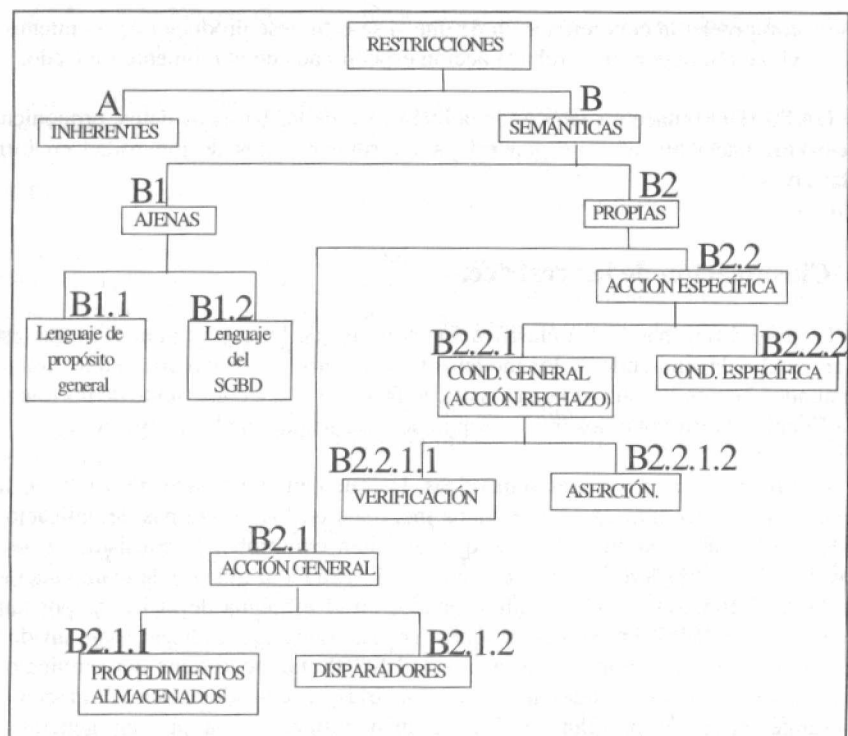


Figura 1.21. Jerarquía de clasificación de las restricciones

Las *restricciones propias del modelo de datos* se especifican al definir el esquema mediante las facilidades que proporciona la función de definición de datos, almacenándose en la base de datos (no en los programas), por lo que no pueden ser violadas por ninguna aplicación, es decir, cualquier actualización está obligada a respetarlas.

Dependiendo de los componentes (acción y/o condición) que haya que especificar al definir una restricción y a la forma de hacerlo (declarativa o procedimental) tendremos distintos tipos de restricciones. En primer lugar, dependiendo de que sea o no preciso definir la acción tendremos restricciones de *acción general* y restricciones de *acción específica*; en las primeras es preciso programar un procedimiento que determine la acción que hay que llevar a cabo, mientras que en las segundas la acción (en general rechazo, aunque puede ser otra, bien predeterminada bien elegida mediante opciones) está implícita en la misma restricción.

Las de acción general son las más flexibles de las restricciones propias del modelo, pero suponen una importante carga de programación; además, el sistema

desconoce su semántica, ya que pueden estar escritas en cualquier lenguaje, por lo que no le es posible comprobar su consistencia ni tampoco el optimizador puede tenerlas en cuenta a fin de mejorar el acceso físico. Son por tanto muy parecidas, en estos aspectos, a las restricciones ajenas al modelo; sin embargo, se diferencian de ellas en que están almacenadas en el esquema, en que su descripción se realiza en el momento de definir el esquema y, principalmente, en que no pueden ser violadas por los programas de aplicación. Se dividen a su vez en:

- *Procedimientos almacenados*

Se definen totalmente de forma procedimental (tanto la acción como la condición); son, entre todas las restricciones propias, las que más se asemejan a las ajenas al modelo⁵⁰.

- *Restricciones de disparo*

Los disparadores⁵¹ permiten definir restricciones de integridad, a las que hemos llamado *restricciones de disparo*. En ellas se formula una condición de forma declarativa, mediante una proposición lógica; el *cumplimiento* de la misma "dispara" una acción especificada de forma procedimental; es decir, al contrario de lo que pasa en otros tipos de restricciones, la acción se desencadena ante un resultado de *cierto* en la condición⁵². La acción ha de programarse mediante un procedimiento, lo que proporciona bastante flexibilidad. Son una mezcla de ambos enfoques declarativo (en la formulación de la condición) y procedimental (en la especificación de la acción).

En las *restricciones de acción específica*, la acción (que puede ser de rechazo o de otro tipo) está determinada por la misma restricción. Son totalmente declarativas porque la acción no hay que definirla y la condición, en el caso de que haya que especificarla, se define de forma declarativa. Dentro de ellas es preciso distinguir:

- *Restricciones de condición general* (se denominan a veces *restricciones generales*⁵³).

La condición se define mediante una proposición lógica, por lo que su complejidad es arbitraria (dentro de lo que permite la proposición) proporcionando, por tanto, una mayor flexibilidad que las restricciones de condición específica que analizaremos posteriormente. La operación será

⁵⁰ Hemos dudado mucho si incluirlas entre las restricciones ajenas al modelo. El haberlas clasificado por fin como restricciones propias se debe, como acabamos de indicar, al hecho de que se definen en el esquema y, lo que consideramos aún más importante, que no pueden ser violadas por los programas.

⁵¹ Los disparadores (*triggers*) son también instrumentos de las bases de datos activas que permiten definir reglas distintas de las restricciones; en realidad, las restricciones, como ya hemos indicado, no son otra cosa que un tipo especial de reglas de las bases de datos activas en las que el evento que las activa es una actualización.

⁵² Si no se especifica condición (la condición es opcional) se considera el resultado como cierto y la acción se dispara siempre que tiene lugar la operación.

⁵³ Dentro de lo que algunos autores denominan *restricciones generales* se considera también a veces los disparadores.

cualquiera que implique asignar valores a los atributos que aparecen en la condición, es decir, una actualización. No se declara la acción porque este tipo de restricción lleva siempre asociado el *rechazo* de la operación cuando no se cumple la condición, es decir, el sistema evalúa la condición y si el resultado es cierto se actualiza, y si no es cierto, no se lleva a cabo la operación.

El SQL 92 distingue dos tipos de restricciones que son de condición general (definida mediante una expresión lógica) y de acción específica (rechazo), que consideramos de interés incluir en esta clasificación:

– *Restricciones de verificación*

Son las cláusulas “CHECK” de algunos lenguajes. La expresión lógica mediante la cual se formula la condición está definida sobre uno o varios atributos de un mismo elemento (por ejemplo, una cierta tabla o un determinado dominio), cuyos valores (o cuyos cambios) han de atenerse a lo especificado en dicha expresión. Este tipo de restricciones se declara al tiempo que se define el elemento del esquema al cual afecta; por ejemplo, si el estado civil de toda persona menor de 14 años tiene que ser soltero, se ha de incluir una restricción de verificación que compruebe que, para persona, “Edad < 14 años y Estado_Civil = Soltero”. Puede dárseles un nombre, pero, al no tener existencia por sí mismas sino dentro del elemento al que afectan, el nombre no es obligatorio.

– *Restricciones de aserción*

Son análogas a las anteriores, aunque se diferencian de ellas en que pueden estar referidas a más de un elemento del esquema (por ejemplo, varias tablas), ya que tienen existencia por sí mismas; por tanto, exigen un nombre.

• *Restricciones de condición específica.*

Reglas de “caso especial” en palabras de CODD (1993) y restricciones “implícitas” para ELSMARI (1989), aunque no siempre está muy clara la distinción que éste hace entre restricciones implícitas y explícitas. Los distintos modelos de datos, cuando se definen los elementos de su esquema, facilitan opciones que son en realidad restricciones; por ejemplo, el modelo relacional, al definir una tabla, da la opción de decir que un atributo o un conjunto de atributos de la misma constituyen una clave primaria, lo que lleva consigo la prohibición de que en la base de datos dos filas de una tabla tengan el mismo valor para estos atributos. Otra restricción específica del modelo relacional es la definición de clave ajena; en este caso, la acción, ante un intento de violación por borrado o modificación de una tupla, la determinará el usuario eligiendo alguna de las opciones que se le ofrecen (según el estándar SQL92, impedir la operación –NO ACTION–, borrar o modificar en cascada –CASCADE–, etc.). Estas restricciones, propias de cada modelo, se declaran

directamente en el esquema mediante opciones que permite el modelo; su intento de violación por una operación de actualización (inserción, borrado o modificación) da como resultado una acción bien determinada⁵⁴. Las hemos llamado restricciones *específicas* para diferenciarlas de todas aquellas en las cuales se declara de forma general la condición o la acción de la restricción; por tanto, estas restricciones se caracterizan porque en ellas no se especifica ninguno de los componentes de una restricción, no existiendo la posibilidad de declarar una condición de tipo general, aunque sí es posible elegir entre opciones que ofrece la misma restricción.

Se podría considerar, como hace ELSMARI (1989), que son únicamente las restricciones específicas las que forman parte de la definición del esquema, y que para las restricciones generales, como propone DATE (1990), debiera existir un lenguaje de definición independiente del modelo, que formaría parte del subsistema de gestión de integridad.

En el cuadro que aparece en el anexo a este capítulo se resume la caracterización de las distintas categorías de restricciones definidas anteriormente.

Existe otra clasificación de las restricciones, conceptualmente ortogonal con la anterior, que las divide en restricciones de *estado* y restricciones de *transición*. En general, las restricciones se aplican a un determinado estado de una base de datos y no hay necesidad de conocer los estados anteriores para saber si se cumple o no la condición, se trata de las restricciones denominadas de *estado*; por ejemplo, la restricción que obliga a que con una edad menor de 14 años el estado civil sea soltero, es una restricción de estado (también llamada *estática*), ya que sólo es necesario comprobar qué ocurre, en ese momento, en la base de datos sin tener en cuenta estados anteriores. Sin embargo, hay veces en que la restricción es de *transición* (o *dinámica*) porque hay que aplicarla a la transición entre dos estados; por ejemplo, el cambio de estado civil, o la que impone que el salario de un empleado no puede disminuir.

A veces se dividen también las restricciones entre aquellas que afectan a un único ejemplar (como "Edad < 14 y Estado_Civil = Soltero") o a más de un ejemplar (como que el sueldo de un empleado del departamento tiene que ser menor que el de su jefe). Dentro de estas últimas es posible distinguir aquellas que sólo afectan a algunos ejemplares y las que afectan a todos los ejemplares de un cierto tipo, en general aplicando alguna medida estadística como un promedio; por ejemplo, que la media de los sueldos de todos los empleados tiene que ser inferior a un cierto valor.

⁵⁴ La acción, por lo general, es un rechazo de la operación, pero también puede ser otra distinta (por ejemplo, el borrado de otras tuplas), dándose en algunas de estas restricciones la posibilidad de optar entre varias acciones siempre bien determinadas.

También se puede distinguir entre las *restricciones de valor* y las *restricciones estructurales*. Las primeras son todas aquellas en las que en la condición se comparan los valores que pueden tomar las propiedades (casi todos los ejemplos que hemos puesto son relativos a este tipo de restricción); existen, además de éstas, otras restricciones que imponen limitaciones a la estructura de los elementos del modelo⁵⁵ (como que un atributo no puede tomar más que un valor o la de que uno o más atributos constituyen la clave primaria⁵⁶), son las *restricciones estructurales*.

En otros casos, se clasifican las restricciones atendiendo a los elementos del modelo de datos a los cuales afecta la condición; así tendríamos restricciones sobre dominios, o sobre relaciones, etc. Esta clasificación es propia de cada modelo de datos, ya que los elementos de los modelos varían de unos a otros.

6. LOS MODELOS DE DATOS EN EL PROCESO DE DISEÑO DE UNA BASE DE DATOS

Se conoce como proceso de diseño de una base de datos al conjunto de etapas necesarias para pasar de una determinada realidad (Universo del Discurso) a la base de datos que la representa. Los modelos de datos desempeñan un importante papel en el proceso de diseño de una base de datos al ofrecernos facilidades de abstracción que nos ayudan a representar la realidad.

Los objetivos que persigue todo modelo de datos son de dos tipos:

- a) **Formalización**, ya que el modelo de datos permite definir formalmente las estructuras permitidas y las restricciones; también el modelo de datos establece la base para la definición de un lenguaje de datos y facilita una apreciación más objetiva de la rigidez o flexibilidad de las estructuras de datos, ayudando a la comparación formal de distintos modelos de datos y a la evaluación de los SGBD.
- b) **Diseño**, ya que el modelo de datos es un elemento fundamental en el desarrollo de una metodología de diseño de bases de datos, en el cual se basan los otros componentes de la metodología (lenguajes, documentación y otras herramientas); permiten, además, prever el impacto de los cambios del mundo real en nuestro sistema de información.

⁵⁵ No confundir con las restricciones inherentes, en las cuales las limitaciones de las estructuras no vienen impuestas por el mundo real sino por el mismo modelo.

⁵⁶ La exigencia de un modelo de la obligatoriedad de la clave primaria es una restricción inherente, que no hay que confundir con la declaración de que uno o más atributos constituyen la clave primaria que es una restricción semántica específica.

La magnitud de la distancia que separa el mundo real de las estructuras de datos almacenadas en los soportes físicos de un computador hace aconsejable abordar el proceso de diseño de una base de datos dividiéndolo en una serie de etapas sucesivas, cada una de las cuales se apoyará en un tipo distinto de modelo de datos adecuado a la correspondiente fase del diseño. Antes de presentar estas etapas hemos de insistir en el concepto de Universo del Discurso (UD) como *visión* que del mundo real tiene el diseñador (véase figura 1.22).

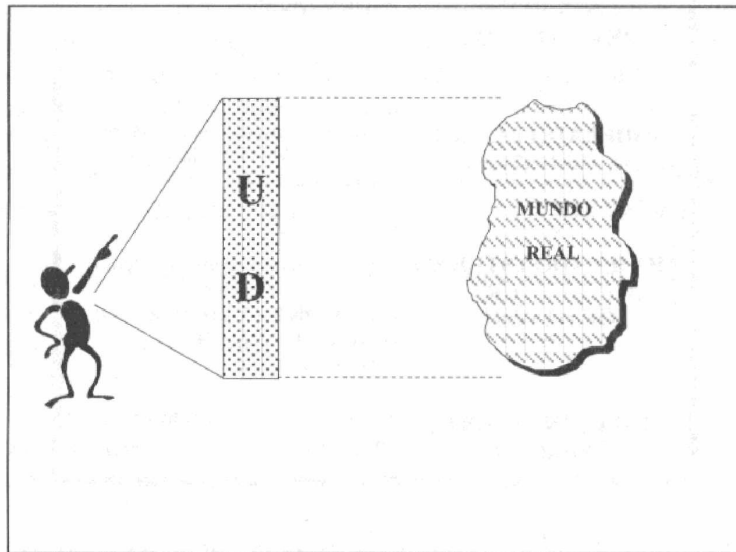


Figura 1.22. Universo del discurso y mundo real

El primer paso en la concepción de una base de datos es definir el universo del discurso, fijando para ello una serie de objetivos sobre el mundo real que se va a analizar; así, por ejemplo, de un mismo mundo real, como puede ser el que constituye una universidad, podemos definir universos del discurso tan distintos como uno relativo a los cursos de doctorado, con los profesores que los imparten, sus departamentos y áreas, los estudiantes, etc.; y otro, concerniente a la gestión de los empleados de la universidad (tanto docentes como no docentes), nóminas, contabilidad, facturación, etc. Es decir, el mundo real es el mismo, pero nuestro objetivo en el primer caso es la docencia de tercer ciclo, mientras que en el segundo es la gestión económica y de personal de la universidad.

Una vez definido el universo del discurso acerca del cual deseamos recoger información en nuestra base de datos, hemos de proceder a su estructuración, paso a paso, hasta llegar a la base de datos física tal como se muestra en la figura 1.23.



Figura 1.23. Etapas en el diseño de una base de datos y tipos de modelos en los que se apoyan

En el diseño de una base de datos es conveniente distinguir la fase de modelado conceptual, que es la descripción del mundo real (empresa o administración) de acuerdo con un modelo conceptual (en la metodología que proponemos será el modelo Entidad/Interrelación –E/R–). El modelo conceptual deberá ser altamente semántico e independiente del SGBD en el que posteriormente se vaya a realizar la implementación de la base de datos, y también independiente de la fase de diseño lógico en la cual se ha de obtener un esquema lógico que responda a la estructura específica (por ejemplo, relacional) del SGBD que se aplique en cada caso concreto; esquema que está sometido a las restricciones que imponga el correspondiente modelo. Posteriormente, el diseño interno permitirá obtener el esquema interno y, por último, se implementará la base de datos física en los soportes secundarios.

Los modelos de datos soportados por los SGBD (Jerárquico, Codasyl y, en la actualidad, principalmente Relacional) no tienen el suficiente poder expresivo para captar la semántica del mundo real que al usuario le gustaría ver plasmada en su base de datos, y sus conceptos, al estar orientados hacia el computador, no suelen ser fácilmente comprendidos por los usuarios. Por estas razones surgen modelos más semánticos y cercanos al usuario, como el modelo Entidad/Interrelación (CHEN,

1976). TSICHRITZIS (1982) insiste en que la forma en que los modelos de datos que hemos llamado convencionales permiten expresar los requisitos de información no se corresponden con la forma en que las personas en general deberían percibir la información representada en la base de datos, y que el papel de los modelos de datos en el diseño de las bases de datos no se debe limitar a abstraer las propiedades de la futura base de datos, sino que ha de servir también como un medio de comunicación entre el usuario y el diseñador. Un modelo de datos cuyos constructores estén basados en el modo en el que las personas perciben los hechos y se los comunican asegura que las estructuras impuestas por el modelo de datos no entrarán en conflicto con las estructuras tal como las percibe el ser humano, es decir, serán modelos orientados a representar la **información** con todo su contenido semántico sin estar “contaminada” por conceptos cercanos al computador, por lo que a estos modelos se los llama **infológicos** (LANGEFORS, 1980 y SUNDGREN, 1974). En cambio, los modelos convencionales están enfocados a representar los **datos**, de acuerdo a cómo han de ser interpretados por los SGBD, recibiendo el nombre de modelos **datológicos**.

El problema se encuentra en que los modelos conceptuales no suelen estar implementados en los SGBD, por lo que los SGBD no “entienden” la estructura conceptual, teniéndose que transformar ésta en una estructura lógica adaptada a las exigencias y restricciones del modelo de datos propio del SGBD que vaya a ser utilizado⁵⁷. Es decir, se debe llegar a un esquema lógico admitido por el SGBD, obteniendo posteriormente el esquema interno, en donde el objetivo es conseguir la máxima eficiencia de cara a la máquina y al problema concreto.

A veces se prescinde de la etapa de modelado conceptual, y el diseñador, sin una metodología precisa, hace una abstracción del mundo real, representándolo directamente en las estructuras del SGBD concreto que va a utilizar; forma de actuar que no consideramos aconsejable, ya que la aplicación de una rigurosa metodología de desarrollo de bases de datos⁵⁸, basada en su primera etapa en un modelo conceptual, ayuda a conseguir una mejor representación del mundo real.

La estructura física resultante del proceso de diseño se ha de *rellenar* con los valores (ejemplares) que se obtienen por observación de los sucesos del mundo real. Estas *cadenas de bits* estarían totalmente carentes de significado si no dispusiéramos de los medios que nos permiten recorrer el camino inverso, pasando de nuevo al mundo real con ayuda del lenguaje de manipulación, por medio del cual actualizaremos o recuperaremos los datos almacenados en la base, reincorporándoles su contenido semántico y obteniendo la información que necesita el usuario.

⁵⁷ Muchas herramientas CASE (*Computer Aided Software Engineering*) pueden realizar de forma automática la transformación de un esquema conceptual en la estructura interna de los SGBD comerciales más extendidos.

⁵⁸ Incluida, en muchos casos, en una metodología general de desarrollo de sistemas de información.

ANEXO. CLASIFICACIÓN DE LAS RESTRICCIONES

A) Inherentes

- Impuestas por el modelo
- No tienen que ser definidas por el usuario, ya que se encuentran en el propio modelo
- Se activan en el momento de la definición del esquema cuando se produce un intento de violación
- Se rechaza todo esquema que no cumple estas restricciones
- Introducen rigideces en el modelo

B) Semánticas

- Impuestas por el universo del discurso
- Tienen que ser definidas por los usuarios (diseñadores)
- Se activan en el momento de la actualización de la base de datos
- Se rechaza todo ejemplar que no cumpla estas restricciones (o se ponen en marcha otros medios a fin de que no se produzca un estado inconsistente)
- Ayudan a capturar la semántica de los datos y a conseguir su consistencia

B1) AJENAS

- Se especifican en los programas de aplicación
- No están almacenadas en el esquema de la base de datos
- Pueden ser violadas por actualizaciones en las que no se haya programado la restricción.
- El SGBD no puede comprobar si son consistentes en sí mismas
- El optimizador no puede tomarlas en consideración
- Proporcionan el máximo de flexibilidad
- Pueden ser programadas en un lenguaje de propósito general o en algún lenguaje propio del SGBD
- Suponen una importante carga de programación y de mantenimiento

B2) PROPIAS

- Se especifican en el esquema
- Están almacenadas en el esquema de la base de datos
- No pueden ser violadas por ninguna actualización

B 2.1) Acción general

- Es obligatorio especificar la condición y la acción
- Son procedimentales (al menos en parte, ya que la acción se especifica siempre mediante un procedimiento)
- Suponen carga de programación
- Es muy difícil (prácticamente imposible en la mayor parte de los casos) que el SGBD pueda comprobar su consistencia
- El optimizador no puede tomarlas en consideración
- Hasta ahora no están estandarizadas⁵⁹
- Están muy ligadas a los productos
- Son muy flexibles
- Tienen nombre y existencia propia dentro del programa

B 2.1.1) Procedimientos almacenados

- Es obligatorio especificar la condición (además de la acción)
- Son totalmente procedimentales
- Pueden ser tan complejas como imponga la semántica del mundo real (tanto en la condición como en la acción)
- Son las más flexibles dentro de las restricciones propias

B 2.1.2) Disparadores

- Combinan los enfoques declarativo (en la condición) y procedimental (en la acción)
- Pueden ser tan complejas como imponga la semántica del mundo real en cuanto a la acción, y bastante complejas en la condición (todo lo que permite la proposición lógica mediante la que se expresa la condición)
- El cumplimiento de la condición (la evaluación de la proposición con resultado de “cierto”) dispara la acción⁶⁰.
- Son más flexibles que las restricciones de acción específica

B 2.2) Acción específica

- La acción está implícita en la misma restricción, por lo que no hay que definirla (está determinada)

⁵⁹ En el estándar conocido como SQL:1999, que será aprobado este año, ya se estandarizan los disparadores.

⁶⁰ En las de acción específica ocurre lo contrario: la acción se dispara cuando la condición no se cumple. En los procedimientos almacenados “en teoría” puede ocurrir tanto una cosa como otra, dependerá de cómo se proponga el procedimiento (en la práctica, los productos pueden imponer ciertas limitaciones).

- Son declarativas, puesto que no se especifica la acción y la condición, si se define, es de forma declarativa
- El no cumplimiento de la condición (su evaluación con resultado de “no cierto”⁶¹) lleva a aplicar la acción
- Podrían ser definidas mediante un lenguaje de tipo general
- El SGBD puede comprobar si son consistentes en sí mismas
- El optimizador puede tomarlas en consideración
- No suponen carga de programación, sólo de definición

B 2.2.1) Condición general

- No se especifica la acción, que es siempre el rechazo (el no cumplimiento de la condición lleva consigo el rechazo de la actualización)
- Es obligatorio declarar la condición mediante una proposición lógica que permite condiciones de complejidad arbitraria (las que permite la proposición)
- Además de la condición, se puede especificar algún otro componente (en especial el momento)
- Son más flexibles que las de condición específica
- Es más difícil optimizar su ejecución que en el caso de las de condición específica

| B 2.2.1.1) Verificación

- No tienen existencia por sí mismas
- Su definición forma parte de la definición del elemento afectado por la restricción
- Se aplican a un único elemento y aunque pueden afectar a otros, en este caso se complica su definición (por lo que es más adecuado utilizar aserciones)
- Pueden no tener nombre

| B 2.2.1.2) Aserción

- Tienen existencia por sí mismas
- Se definen con independencia de cualquier elemento del esquema (no forma parte de la definición de ningún elemento)
- Pueden afectar a más de un elemento
- Tienen nombre

⁶¹ Decimos “no cierto” en lugar de *falso* porque incluye el “quizás” en el caso de admitirse nulos y ser “quizás” el resultado de la evaluación.

B 2.2.2) De condición específica

- Son opciones proporcionadas por el propio modelo
- No se especifica ninguno de los componentes relativos a una restricción (ni la operación, ni la condición, ni la acción)
- Son poco flexibles
- El optimizador puede tomarlas en consideración
- Su ejecución puede ser más fácilmente optimizada que las de condición general

CAPÍTULO 2

MODELO ENTIDAD/INTERRELACION (ME/R)

En este capítulo analizaremos, en el marco de los modelos expuestos en el capítulo anterior, el modelo Entidad/Interrelación, uno de los modelos conceptuales más extendidos en las metodologías de diseño de bases de datos y en las herramientas CASE. Después de presentar la historia del modelo, estudiaremos su componente estática y el formalismo gráfico asociado a ésta (que ha contribuido sin duda a su éxito), y profundizaremos en otros aspectos introducidos en posteriores extensiones y refinamientos, los cuales, al dotarlo de más elementos semánticos, lo convierten en un instrumento fundamental en el diseño conceptual de bases de datos. Vamos a ocuparnos tan sólo de la parte estática del modelo, ya que la componente dinámica del mismo apenas tiene interés en modelos de este tipo, enfocados al diseño conceptual de bases de datos.

1. PRESENTACIÓN DEL MODELO

Los modelos de datos convencionales no ofrecen la suficiente capacidad de abstracción ni el poder expresivo como para captar la semántica del mundo real, haciendo difícil la comunicación del diseñador con el usuario. Entre los modelos de datos que surgen a fin de paliar estos problemas, destaca el Modelo Entidad/Interrelación (E/R), propuesto por Peter P. Chen en sus dos artículos ya históricos, CHEN (1976) y CHEN (1977). Según CHEN (1976), “El modelo E/R puede ser usado como una base para una vista unificada de los datos”, adoptando “el enfoque más natural del mundo real que consiste en *entidades e interrelaciones*”.

Posteriormente otros muchos autores, PAUL (1980), TEOREY *et al.* (1986), FERG (1984), SHANG y SHIXUAN (1984), ELSMARI *et al.* (1985), etc., han investigado y escrito sobre el modelo, proponiendo importantes aportaciones. Realmente no se puede considerar que exista un único modelo E/R, sino más bien lo que podríamos llamar una *familia de modelos*, por lo que hay importantes diferencias en la presentación que del modelo hacen distintos autores. En el modelo que vamos a exponer se incluyen la mayoría de las extensiones que se han ido aportando a lo largo del tiempo, así como algunas consideraciones e interpretaciones propias, y que creemos clarifican, precisan o amplían algunos conceptos.

El modelo E/R permite al diseñador concebir la base de datos a un nivel superior de abstracción, aislándolo de consideraciones relativas a la máquina (tanto en su nivel lógico como físico) y a los usuarios en particular (nivel externo), y centrándolo en un plano infológico en el que la información desempeña un papel fundamental. El modelo, como su nombre indica, se apoya en dos conceptos: *entidad* e *interrelación*¹. Para CHEN (1976), una entidad es “una cosa que se puede identificar claramente” y “una interrelación una vinculación entre entidades”.

Desde que fue propuesto, el modelo E/R ha tenido una gran difusión y ha despertado un enorme interés en la comunidad informática dedicada a las bases de datos, prueba de ello son los innumerables congresos dedicados al tema, entre los que caben destacar las *Conferencias Internacionales sobre el enfoque E/R* que se celebran anualmente; es también el modelo más extendido en las herramientas CASE de ayuda al diseño de bases de datos. Sin embargo, algunos autores (especialmente DATE en sus diversas obras²) son críticos acérrimos del modelo E/R. DATE critica el modelo E/R basándose en la idea de que no es tan siquiera un modelo de datos, sino más bien “una delgada capa encima del modelo relacional básico”; asimismo, afirma que no es un modelo formal y que los pocos aspectos formales que presenta no son muy diferentes a los correspondientes aspectos del modelo relacional básico. Sin embargo, también reconoce que, a pesar de ser un modelo que no podrá sustituir al modelo relacional, sí se trata de un modelo útil.

Estudiaremos el modelo E/R en el marco general de los modelos de datos que hemos presentado en el capítulo anterior.

¹ Hemos traducido por “interrelación” el término inglés *relationship* a fin de distinguirlo de la “relación” del modelo relacional que, en inglés, se denomina *relation*. No lo hemos querido llamar asociación porque hemos preferido reservar este vocablo para referirnos al caso general de conexión entre objetos, dejando interrelación para expresar la asociación en el modelo específico que estudiamos en este capítulo. Sin embargo, nos ha parecido más conveniente mantener las siglas E/R por estar muy extendidas.

² Véase, por ejemplo, DATE (1993).

2. ESTÁTICA DEL MODELO E/R

En el modelo E/R, tal como fue propuesto por Chen, se distinguen los siguientes elementos: Entidad, Interrelación, Atributo y Dominio (para Chen *conjunto de valores*³).

2.1. Entidad

Se puede definir una entidad como cualquier objeto⁴ (real o abstracto) que existe en la realidad y acerca del cual queremos almacenar información en la base de datos. HALL (1976) la define como “algo con realidad objetiva que existe o puede ser pensado”. Según ANSI (1977), es “una persona, lugar, cosa, concepto o suceso, real o abstracto, de interés para la empresa”. La estructura genérica que describe un conjunto de entidades aplicando la abstracción de clasificación se denomina *tipo de entidad*, mientras que *entidad* es cada uno de los ejemplares⁵ de ese tipo de entidad; por tanto, el tipo de entidad es el resultado de la *clasificación* de un conjunto de entidades. Así, **CURSO** es un tipo de entidad que describe las características comunes de un conjunto de cursos; un ejemplar del tipo de entidad **CURSO** será, por ejemplo, “Diseño de Bases de Datos Relacionales” y otro “Introducción a los Sistemas de Bases de Datos”. Otro tipo de entidad podría ser **PROFESOR** y un ejemplar del mismo sería Sr. Sánchez.

El conjunto de ejemplares de un tipo de entidad en un momento dado será la extensión de ese tipo de entidad, mientras que la intensión es el tipo de entidad propiamente dicho. Cuando por el contexto se sobreentiende que nos estamos refiriendo a un tipo de entidad, se simplifica a veces la expresión y se utiliza únicamente entidad. Chen habla de *conjunto de entidades* (*entity set*), lo que para él es análogo a tipo de entidad.

Si una entidad pertenece a un tipo de entidad, ha de cumplir el predicado asociado al correspondiente tipo de entidad. Matemáticamente, un conjunto de ejemplares de un tipo de entidad se define entonces como:

$$\{ e : p(e) \}$$

siendo *e* un ejemplar del tipo de entidad *E* y *p* el predicado asociado a *E*. por ejemplo, el tipo de entidad **PROFESOR**, cuyo predicado asociado es “*Persona que ejerce o enseña una materia o arte*” tiene un ejemplar “*Sánchez*” que pertenece a él, ya que cumple dicho predicado.

³ En inglés *values set*.

⁴ Tomamos objeto en el sentido que tiene en el lenguaje común, y no con la acepción específica del paradigma de la orientación al objeto.

⁵ Recuérdese lo que se dijo en el capítulo anterior con respecto a este término.

La representación gráfica de un tipo de entidad en este modelo es un rectángulo etiquetado en cuyo interior está el nombre del tipo de entidad, como podemos ver en la figura 2.1.

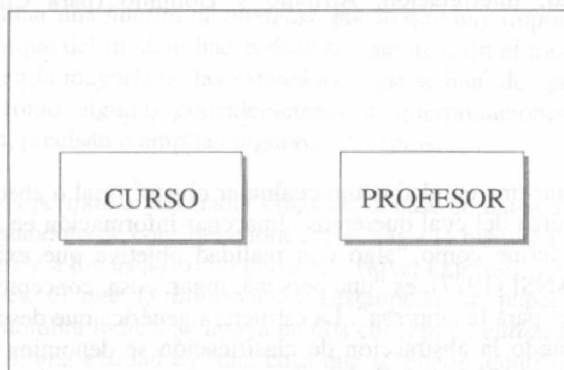


Figura 2.1. Representación de tipos de entidad

Existen dos clases de entidades: *regulares*, que son aquellas cuyos ejemplares tienen existencia por sí mismos (como **CURSO** y **PROFESOR**), y *débiles*, en las cuales la existencia de un ejemplar depende de que exista un cierto ejemplar de otro tipo de entidad (por ejemplo, **EDICIÓN** depende de **CURSO**), y la desaparición de un determinado curso de la base de datos hace que desaparezcan también todas las ediciones de dicho curso.

Los tipos de entidad débil se representan con dos rectángulos concéntricos con su nombre en el interior, como se puede observar en la figura 2.2.

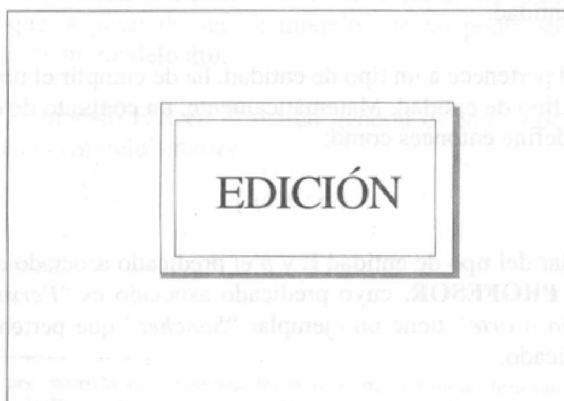


Figura 2.2. Representación de una entidad débil

Aunque es fácil entender el concepto de entidad, no lo es su definición formal; por esta razón, se ha afirmado a veces que es preferible dejar el término sin definir. En nuestra opinión, el problema, más que en la definición en sí misma, se encuentra en que un cierto objeto del mundo real se cataloga en ocasiones como una entidad, mientras que en otras se considera una propiedad de una entidad o una interrelación; un ejemplo muy repetido de ello es el *color*, el cual es en general una propiedad de una entidad (como es el caso del color de un coche); sin embargo, en una fábrica de pinturas probablemente sería apropiado modelar el color como una entidad que tendría sus propiedades.

Algunos autores han intentado precisar el concepto de entidad. Así, TARDIEU *et al.* (1979) propone tres reglas generales que debe cumplir una entidad:

- Tiene que tener existencia propia.
- Cada ejemplar de un tipo de entidad debe poder distinguirse de los demás.
- Todos los ejemplares de un tipo de entidad deben tener las mismas propiedades.

Sin embargo, la primera de estas reglas no es aplicable a las entidades débiles, cuya existencia depende de la existencia de la entidad regular de la cual dependen. En cuanto a la segunda de estas condiciones supone la obligación de un identificador que permita distinguir los distintos ejemplares de un tipo de entidad, lo que tampoco es universalmente aceptado (ni por los autores, ni por los modelos, ni por los productos). Respecto a la tercera: ¿hasta qué punto todos los ejemplares de un tipo de entidad tienen las mismas propiedades en el caso de que el modelo admita valores nulos (especialmente los inaplicables⁶)?

2.2. Interrelación

Se entiende por *interrelación* una asociación, vinculación o correspondencia entre entidades. Denominaremos *tipo de interrelación* a la estructura genérica que describe un conjunto de interrelaciones, mientras que interrelación será cada uno de los ejemplares concretos; por tanto, el tipo de interrelación es el resultado de *clasificar* un conjunto de interrelaciones. Por ejemplo, *Imparte* es un tipo de interrelación que vincula los dos tipos de entidad **PROFESOR** y **CURSO**; un ejemplar del tipo de interrelación *Imparte* es la vinculación entre el profesor Sr. Sánchez y el curso “Diseño de Bases de Datos Relacionales”.

Matemáticamente, el conjunto de interrelaciones de un tipo de interrelación *I* se define como:

⁶ El lector que no conozca los conceptos de valor nulo e inaplicable puede consultar la obra DE MIGUEL y PIATTINI (1999)

$$\{ \langle e_1, e_2, \dots, e_n \rangle \}$$

donde e_i es un ejemplar del tipo entidad E_i y n el grado del tipo de interrelación, es decir, el número de tipos de entidad que están asociados en el tipo de interrelación.

Representaremos el tipo de interrelación mediante un rombo etiquetado con el nombre de la interrelación, unido mediante arcos a los tipos de entidad que asocia, como se puede observar en la figura 2.3, en donde establecemos la interrelación **Imparte** entre **PROFESOR** y **CURSO**.

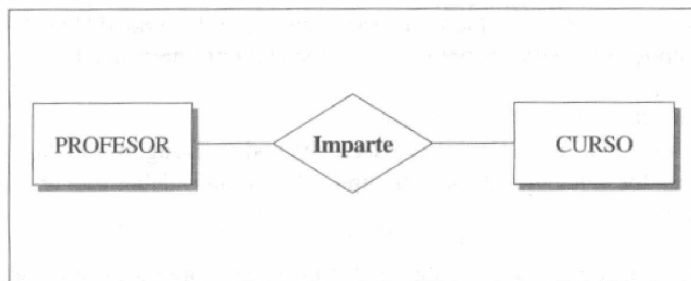


Figura 2.3. Representación de la relación **Imparte** entre **PROFESOR** y **CURSO**

Entre dos tipos de entidad puede existir más de un tipo de interrelación, como se puede observar en la figura 2.4.

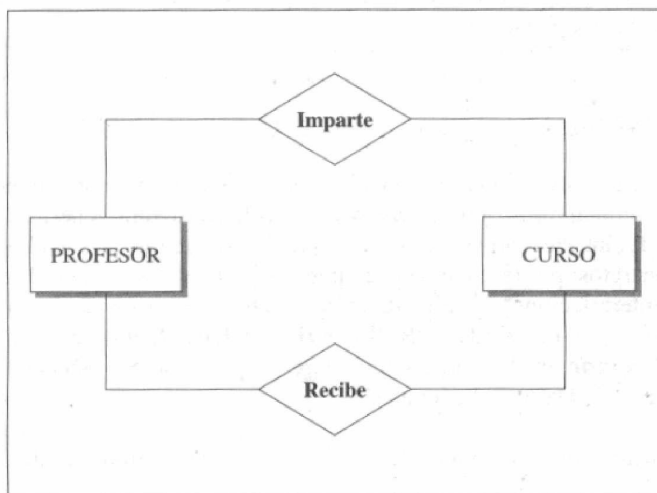


Figura 2.4. Dos tipos de entidad entre los que existen dos tipos de interrelación

Varios autores no están de acuerdo en que se distinga entre entidades e interrelaciones. Así, para DATE (1993), tal distinción no tiene sentido ya que un mismo objeto del mundo real puede ser visto como entidad o como interrelación; todo depende del dominio de la aplicación. Para Date una interrelación es un tipo especial de entidad. Un ejemplo es el matrimonio, que puede ser visto como una interrelación entre dos personas o como una entidad en sí misma. Esta misma idea se recogía ya en HSU y ROUSSOPOULUS (1980). Para estos autores, las interrelaciones son entidades y difieren de éstas en que, mientras las entidades tienen propiedades propias, las interrelaciones tienen propiedades de otros objetos.

En nuestra opinión, la interrelación puede considerarse un tipo especial de entidad cuya existencia depende de la existencia de las entidades a las que relaciona⁷, pero su especificidad hace necesario definirla como un elemento del modelo de datos distinto de la entidad; por otro lado, discrepamos de Date, ya que creemos que puede tener atributos propios, precisamente porque es un tipo especial de entidad. Aunque es cierto que un mismo objeto puede ser visto como una entidad o como una interrelación y que dependerá del universo del discurso que se esté analizando (al igual que ocurre, como ya se ha comentado, con los atributos y las entidades), ello no tiene por qué llevar necesariamente a la conclusión de que no se debe distinguir entre entidades e interrelaciones.

2.3. Dominio y valor

Las distintas propiedades o características de un tipo de entidad o de interrelación toman **valores** para cada ejemplar de éstas. El conjunto de posibles valores que puede tomar una cierta característica se denomina **dominio**⁸. Se define dominio como un conjunto de valores homogéneos con un nombre. Para saber si un valor pertenece a un dominio determinado, comprobaremos que cumple el predicado que el dominio lleva siempre asociado. Matemáticamente se expresa:

$$D = \{ v_i : p(v_i) \}$$

donde D es el dominio, v_i es un valor y p es el predicado asociado a dicho dominio.

Por ejemplo, el valor “inglés” se toma del dominio *Idiomas*, y cumple el predicado de ser uno de los idiomas posibles del conjunto (“español”, “inglés”, “francés”); el dominio **Nombre_curso** es una tira de caracteres. Un dominio puede definirse por *intensión*, especificando el tipo de datos (por ejemplo, carácter 30 para el **Nombre_curso** o fecha para la **Fecha_edición**); o por *extensión*, declarando el valor de cada elemento del dominio (como es el caso de Idioma). El dominio se representa

⁷ Un análisis profundo de la distinción entre entidades e interrelaciones puede encontrarse en MARCOS (1997).

⁸ Aunque, como ya hemos advertido, Chen en la presentación del modelo E/R emplea la expresión “Conjunto de valores” nosotros hemos optado por el término dominio propio del modelo relacional, que está mucho más extendido.

gráficamente con un círculo u óvalo etiquetado con su nombre. En la figura 2.5 mostramos dos formas de representar un dominio.

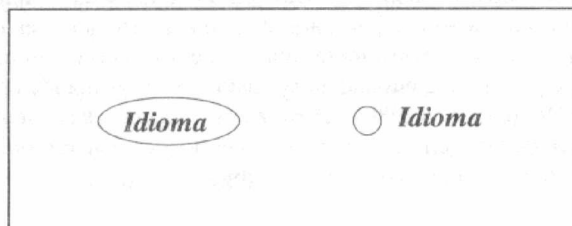


Figura 2.5. Dos formas de representación de un dominio

El dominio es un elemento del modelo que tiene existencia propia independiente de cualquier otro elemento.

2.4. Atributo

Cada una de las propiedades o características que tiene un tipo de entidad o un tipo de interrelación se denomina **atributo**; los atributos toman valores de uno o varios dominios. Por tanto, podemos decir que el atributo le da una determinada interpretación al dominio (o a los dominios) en el contexto de un tipo de entidad o de un tipo de interrelación. Matemáticamente consiste en una función de un tipo de entidad o de interrelación sobre todos los posibles subconjuntos de los valores de un dominio o de un conjunto de dominios:

$$A : E \rightarrow S(D) \text{ ó } A : E \rightarrow S(D_1) \times S(D_2) \times \dots \times S(D_n)$$

$$A : I \rightarrow S(D) \text{ ó } A : I \rightarrow S(D_1) \times S(D_2) \times \dots \times S(D_n)$$

donde A es el atributo, $S(D_i)$ todos los posibles subconjuntos de los valores de los dominios, E el tipo de entidad e I el tipo de interrelación.

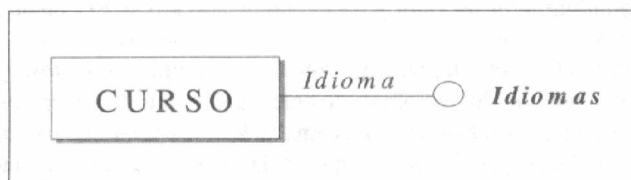


Figura 2.6. Representación de dominio y de atributo

La representación gráfica de un atributo consiste en cualificar con su nombre el arco que une el dominio con el tipo de entidad o de interrelación (ver figura 2.6). Sin

embargo, para simplificar la representación gráfica y siempre que coincida el nombre del dominio con el atributo, será suficiente con el círculo u óvalo con el nombre del atributo.

En el esquema conceptual resultante del modelado sólo especificaremos los atributos más significativos; en la figura 2.7. se representan los tipos de entidad **CURSO** y **PROFESOR** y el tipo de interrelación **Imparte** con alguno de sus atributos.

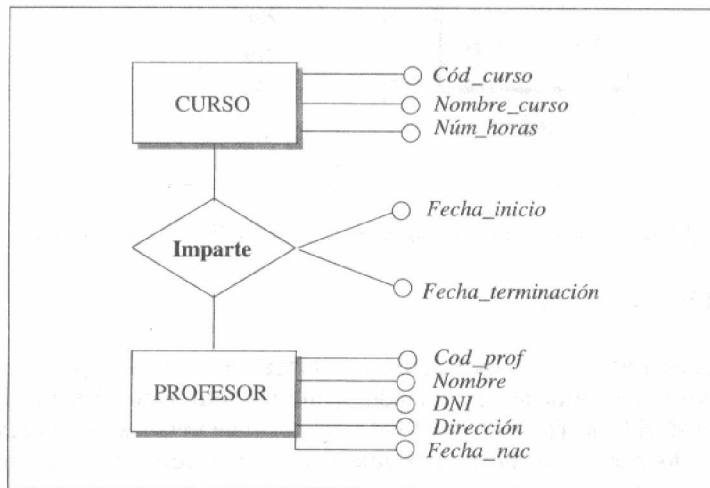


Figura 2.7. Representación de atributos de tipos de entidades y de interrelación

El modelo E/R admite —como se deduce de la definición de atributo— atributos compuestos, es decir, atributos definidos sobre más de un dominio; por ejemplo, el atributo *Fecha_Nac* de la entidad **PROFESOR** puede estar definido sobre los dominios *Día*, *Mes*, y *Año*. En la figura 2.8 se muestran dos formas de representar los atributos compuestos

A diferencia de los dominios que tienen vida propia, es decir, existen por sí mismos, la existencia de un atributo está ligada a la del correspondiente tipo de entidad. Así la fecha de nacimiento de un empleado (*Fecha_Nac*) no tiene sentido si de nuestro esquema desaparece el tipo de entidad **EMPLEADO**; sin embargo, el dominio *Fechas* puede existir con independencia de cualquier otro tipo de entidad o atributo.

Por otro lado, debemos observar que mientras los tipos de entidad tienen atributos, sus ejemplares toman valores para cada atributo, aunque a veces, a fin de simplificar, se hable de forma poco precisa de los atributos de una entidad.

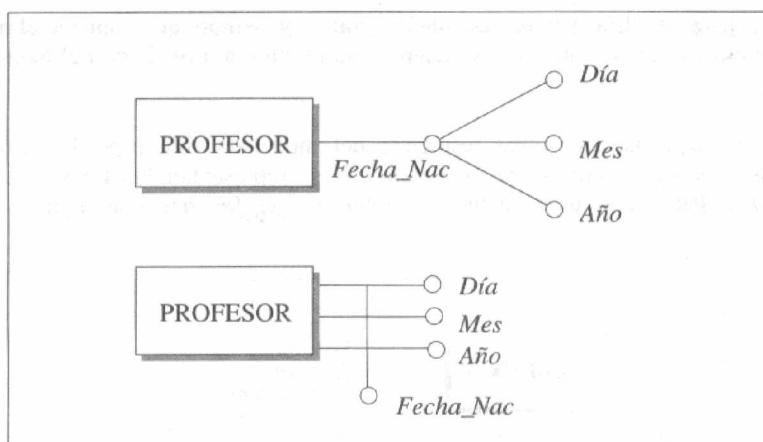


Figura 2.8. Dos formas de representar atributos compuestos

3. RESTRICCIONES

El modelo E/R tiene como restricción inherente que sólo permite establecer interrelaciones entre entidades, no estando admitidas entre entidades e interrelaciones ni entre interrelaciones. También obliga el modelo a que todas las entidades tengan un identificador, lo que asimismo podría considerarse una restricción inherente. El no tener apenas restricciones inherentes dota al modelo de una gran flexibilidad para la representación del mundo real.

En cuanto a restricciones de integridad, únicamente consideramos las restricciones específicas, distinguiendo entre las restricciones sobre valores y las estructurales.

Las restricciones sobre valores se establecen mediante la definición de dominio, la cual permite limitar los valores del dominio y, por tanto, los de los atributos sobre él definidos, a los de un determinado tipo de datos, o restringirlos a los comprendidos en un rango, o bien declarar los valores posibles en el caso de que la definición se haga por extensión.

Las restricciones estructurales se refieren tanto a atributos como a interrelaciones; estas últimas las analizaremos más adelante cuando tratemos la semántica de las interrelaciones, mientras que de las que atañen a los atributos nos ocupamos a continuación.

Entre todos los atributos de un tipo de entidad han de existir uno o varios (simples y/o compuestos) que identifiquen unívocamente cada una de los ejemplares de ese tipo de entidad. Cada uno de estos conjuntos de atributos se denomina *Identificador*

Candidato (IC). Cuando un IC es compuesto, el número de los atributos que lo componen debe ser mínimo, en el sentido de que la eliminación de cualquiera de ellos le haría perder su carácter identificador. Luego todo IC debe cumplir la condición de ser unívoco y mínimo. Entre los IC se elige uno como *Identificador Principal* (IP) y el resto serán *Identificadores Alternativos* (IA). La representación gráfica de estos atributos queda reflejada en la figura 2.9.

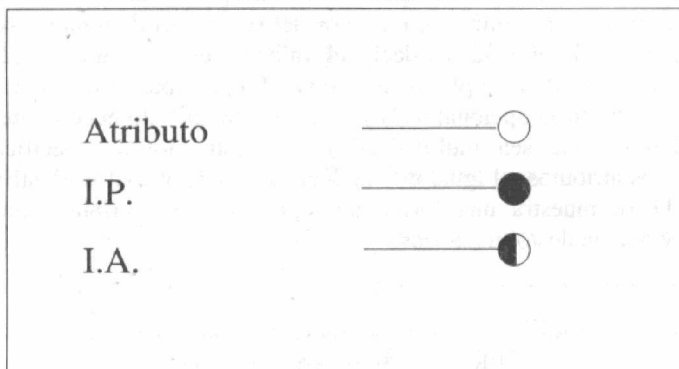


Figura 2.9. Representación gráfica de IP y IA

Los identificadores principales (o alternativos) compuestos se pueden representar de forma análoga a la de los atributos compuestos tal como se muestra en el ejemplo de la figura 2.10.

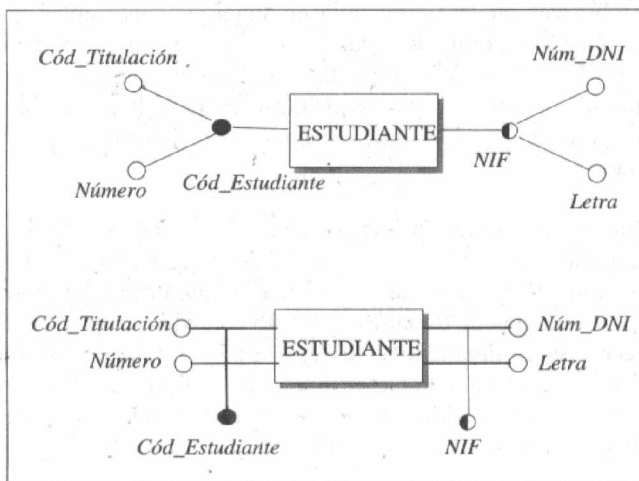


Figura 2.10. Ejemplo de IP y de IA compuestos

El modelo E/R permite también atributos *multivaluados* y *opcionales* (nulos o “faltantes”). En general un atributo toma, para cada ejemplar de entidad, un único valor de cada dominio (o dominios) subyacente(s) (un libro tiene un único título, un único ISBN, etc.), pero también existen atributos que pueden tomar más de un valor (un curso puede impartirse en más de un idioma, o un profesor puede tener más de un teléfono); estos atributos reciben el nombre de *multivaluados* frente a los *univaluados* que toman un solo valor. Por otro lado, puede obligarse a un atributo de un tipo de entidad a que tome, como mínimo, un valor del (o de los) dominio(s) subyacente(s) para cada ejemplar de entidad; es decir, el valor de ese atributo es *obligatorio* (no puede ser nulo) para todo ejemplar de la entidad. La prohibición de valores nulos para un atributo (no admitir la opcionalidad) y la de que un atributo pueda tomar más de un valor (no admitir que sea multivaluado) son restricciones específicas sobre la estructura de los atributos, al igual que la declaración de atributos identificadores. En la figura 2.11 se muestra una forma de representar los atributos multivaluados/univaluados y opcionales/obligatorios.

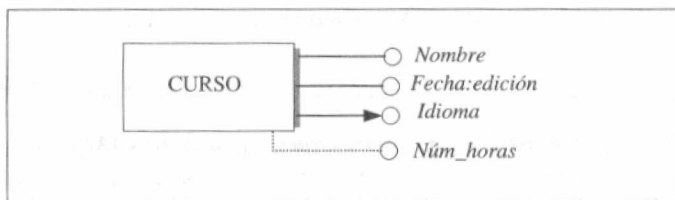


Figura 2.11. Ejemplo de atributos multivaluado (*Idioma*) y opcional (*Núm_Horas*)

Se puede observar en la figura que, en lugar de representar la existencia de restricción (univaluación u obligatoriedad de un atributo), lo que se representa con un símbolo especial (línea discontinua o punta de flecha) es la ausencia de restricción; la razón es que lo más habitual es que un atributo sea univaluado y obligatorio, por lo que son éstas las características que se toman por defecto y, por tanto, son las contrarias las que se representan con símbolos especiales.

Todas estas restricciones pueden definirse basándose en el concepto de cardinalidad de un atributo en el tipo de entidad o de interrelación al cual pertenece. Se entiende por cardinalidad mínima (o máxima) de un atributo el número mínimo (o máximo) de valores que puede tomar ese atributo en cada ejemplar del tipo de entidad al cual pertenece; las cardinalidades se representan asociando un par de números enteros (mín, máx) al correspondiente atributo⁹. En la figura 2.12 aparecen los cuatro tipos posibles de cardinalidades, junto con la otra forma de representación que mostrábamos en el ejemplo de la figura 2.11.

⁹ Se ha de observar que las cardinalidades aportan más semántica que la línea de puntos o la punta de flecha cuando se conoce exactamente el número mínimo (o máximo) de valores que puede tomar el atributo en cada ejemplar del tipo de entidad.

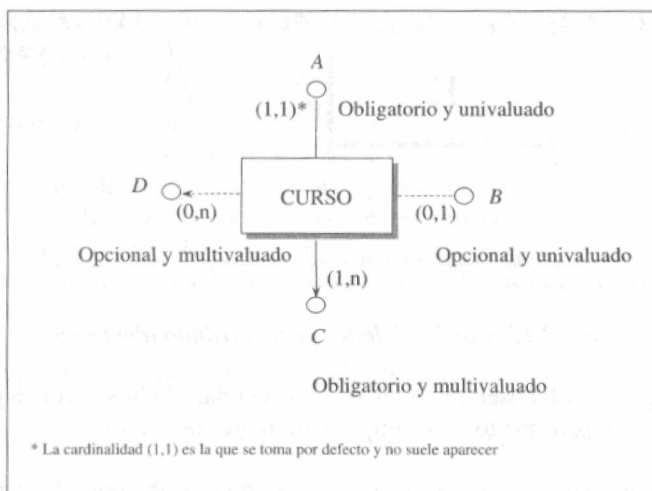


Figura 2.12. Representación de los cuatro tipos posibles de cardinalidades de atributos

También la cardinalidad, pero no del atributo sino del tipo de entidad respecto al atributo, permite representar otra restricción que es la *unicidad*, por la cual se obliga a que los valores de un atributo no puedan repetirse en distintos ejemplares de un tipo entidad, en cuyo caso la cardinalidad máxima de esa entidad respecto al atributo es uno. Debemos observar que para todo identificador de un tipo de entidad se ha de cumplir la restricción de unicidad, debiendo tener el tipo de entidad una cardinalidad máxima de uno respecto a ese atributo. Sin embargo, la recíproca no es cierta, ya que la unicidad de un atributo no implica que sea un identificador, porque si el atributo es compuesto es preciso exigir, además, la condición de minimalidad; y, en todo caso, sea o no compuesto, se debe imponer también que se cumplan las restricciones de obligatoriedad y de univaluación¹⁰. La cardinalidad mínima de la entidad respecto al atributo no tiene sentido, pero sí lo tiene respecto al dominio; un valor de cero de la misma indica que puede haber valores del dominio sobre el cual esté definido el atributo que no aparezcan en el atributo para ningún ejemplar del tipo de entidad, mientras que un valor de 1 indica que todos los valores del dominio deben aparecer como valores del atributo en alguna de las instancias del tipo de entidad. En la figura 2.13 aparecen las cardinalidades del identificador de la entidad E.

¹⁰ La restricción de obligatoriedad de un atributo identificador no se suele tener en cuenta en el modelo E/R, si bien sí se exige en el modelo relacional aunque sólo para la clave primaria. En cuanto a la restricción de univaluación de un atributo identificador, no se hace ninguna referencia en los textos consultados, probablemente porque se da por supuesto; sin embargo, en nuestra opinión, se debe imponer explícitamente.

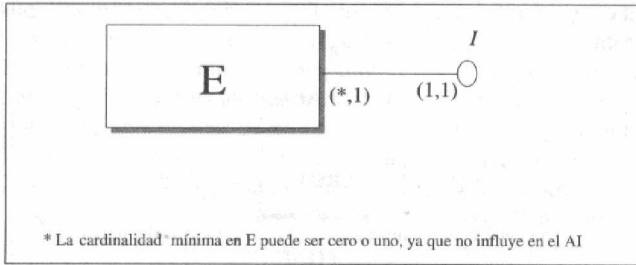


Figura 2.13. Cardinalidades de un atributo identificador

En la figura 2.14 observamos el tipo de entidad CURSO con algunos de sus atributos y un ejemplar que toma valores de diferentes dominios.

Con independencia de las restricciones que acabamos de ver, y de las restricciones estructurales sobre interrelaciones que estudiamos posteriormente (todas ellas restricciones de condición específica), el modelo E/R no proporciona instrumentos para la declaración de otras restricciones (restricciones de condición general), las cuales sólo podrían ser formuladas mediante un lenguaje general de definición de restricciones, ajeno al modelo E/R o por medio de comentarios que acompañen al diagrama.

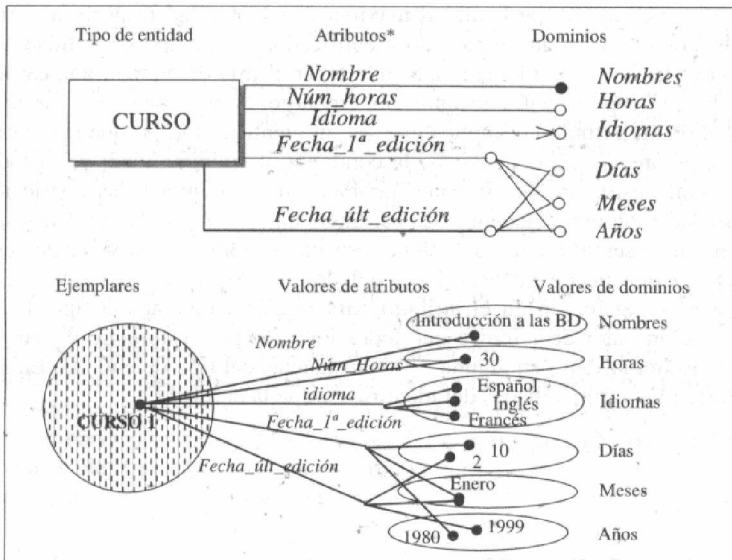


Figura 2.14. Ejemplo del tipo de entidad CURSO, con alguno de sus atributos, y de un ejemplar de CURSO con sus valores

4. PRIMERA APROXIMACIÓN A LA SEMÁNTICA DE LAS INTERRELACIONES

El contenido semántico de las interrelaciones se ha ido completando con conceptos tales como las cardinalidades, la dependencia en existencia y en identificación, la abstracción de generalización, etc.; en este epígrafe vamos a comenzar viendo los elementos de una interrelación que aparecen en el modelo básico así como algunos aspectos semánticos como las dependencias en existencia y en identificación. Posteriormente, en otros epígrafes, iremos extendiendo la semántica de las interrelaciones.

4.1. Elementos de un tipo de interrelación

En un tipo de interrelación se pueden distinguir los siguientes elementos:

- **Nombre:** Al igual que las entidades, los dominios y los atributos, cada tipo de interrelación tiene un nombre que lo distingue unívocamente del resto, y mediante el cual ha de ser referenciado. Como hemos indicado anteriormente, en la representación gráfica del tipo de interrelación (un rombo etiquetado) siempre ha de aparecer el nombre, el cual aporta semántica al modelo; otros modelos de datos (como el jerárquico e incluso el relacional para ciertos tipos de interrelación) no soportan esta semántica.
- **Grado:** Es el número de tipos de entidad que participan en un tipo de interrelación. Así, un tipo de interrelación es de grado 2 (o binaria) cuando asocia dos tipos de entidad como las de las figuras 2.3 y 2.4. Un caso particular de interrelaciones de grado 2 son las *reflexivas* (también llamadas recursivas en algún texto), las cuales asocian un tipo de entidad consigo misma; en la figura 2.15 se muestra el tipo de interrelación reflexiva **Consta** que asocia TEMA con TEMA, en la que se refleja la posibilidad de que un cierto tema (por ejemplo, informática) esté compuesto por (sub)temas (por ejemplo, bases de datos, sistemas operativos, lenguajes, etc.).

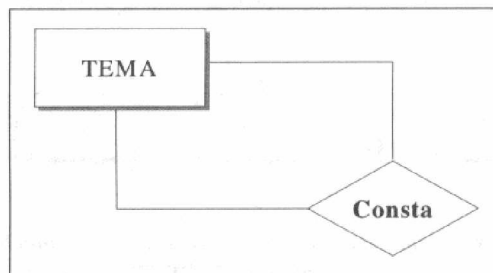


Figura 2.15. Ejemplo de tipo de interrelación reflexiva

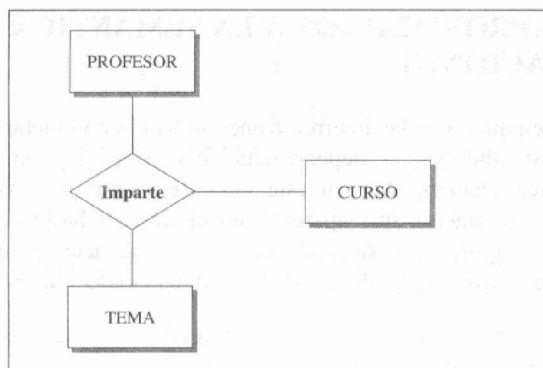


Figura 2.16. Ejemplo de un tipo de interrelación de grado superior a dos

Pueden existir también tipos de interrelación que asocien más de dos tipos de entidad (grado n , $n > 2$) como en la figura 2.16¹¹. En este ejemplo se muestra un profesor con los temas y cursos que imparte.

- **Tipo de correspondencia:** Es el número máximo de ejemplares de un tipo de entidad que pueden estar asociados, en una determinada interrelación, con un ejemplar de otro(s) tipo(s); para representarlo gráficamente, bien se pone una etiqueta con 1:1, 1:N o N:M según corresponda al lado de la interrelación, o bien se orienta el arco de unión en el sentido 1 a N mediante una punta de flecha, tal como aparece en la figura 2.17, donde se han incluido ambos tipos de representación en tres ejemplos de tipos de interrelación.

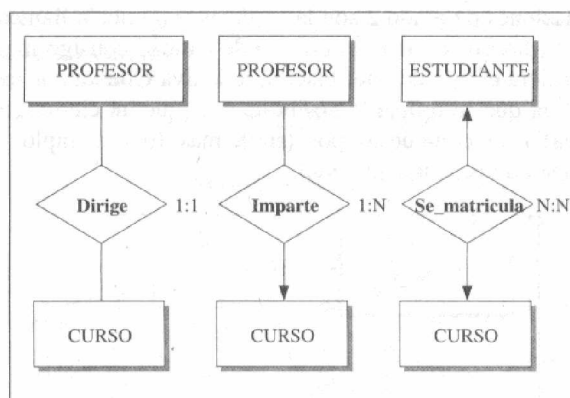


Figura 2.17. Ejemplos de interrelación uno a uno (1:1), uno a muchos (1:N), y muchos a muchos (N:N)

¹¹ La semántica de estas interrelaciones se analizará posteriormente.

- **Papel (“rol”)**: Es la función que cada uno de los tipos de entidad realiza en el tipo de interrelación; se representa poniendo el nombre del **papel** en el arco que une cada tipo de entidad con el tipo de interrelación (ver figura 2.18). Siempre que no exista ambigüedad se suele prescindir de representar el papel.

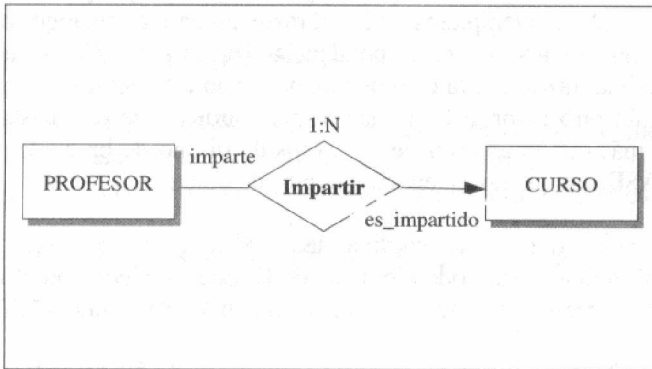


Figura 2.18. Representación de los “papeles” en un tipo de interrelación

4.2. Cardinalidad de un tipo de entidad

Se define como el número máximo y mínimo de ejemplares de un tipo de entidad que pueden estar interrelacionadas con un ejemplar del otro, u otros tipos de entidad que participan en el tipo de interrelación. Se representa gráficamente mediante una etiqueta del tipo (0,1), (1,1), (0,N) ó (1,N), según corresponda, al lado de los tipos de entidades asociados por el tipo de interrelación, tal como aparece en la figura 2.19. El concepto de cardinalidad, tal y como se ha definido aquí, no coincide exactamente con el propuesto en TARDIEU *et al.* (1979), el cual contempla la cardinalidad como el número mínimo y máximo de ejemplares de cada tipo de entidad que intervienen en una interrelación; en el caso de interrelaciones binarias las etiquetas aparecerían intercambiadas de lugar con respecto a nuestra definición, pero si se trata de interrelaciones de grado superior a dos los valores de las cardinalidades cambian porque el concepto es distinto.

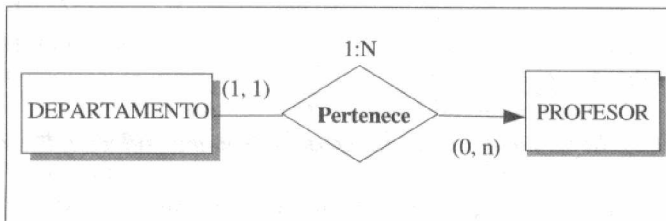


Figura 2.19. Ejemplo de interrelación en la que aparecen las cardinalidades

Sea I un tipo de interrelación binaria y E_1 y E_2 los tipos de entidad asociados por ella. Si no se impone restricción alguna a $I(E_1 \rightarrow E_2)$ y $I(E_2 \rightarrow E_1)$, cualquier número de ejemplares de entidad, ninguno o varios a la vez, de E_1 pueden estar relacionados con uno de E_2 y viceversa. Se utilizará la notación $I(E_1(0,n)):E_2(0,n)$ para denotar esta clase de interrelaciones. En esta notación, $E_1(0,n)$ significa que un ejemplar de E_2 puede estar relacionado con 0,1,2,...,n ejemplares de E_1 ; el razonamiento es análogo para $E_2(0,n)$. Se puede observar que el tipo de correspondencia definido por Chen coincide con la cardinalidad máxima, razón por la cual hemos preferido esta notación¹², que es también utilizada por algún otro autor, a la propuesta por Tardieu que está bastante extendida (probablemente más que la anterior en los libros de diseño de bases de datos y en las herramientas CASE, aunque no así en los modelos de objetos).

Una aplicación donde la cardinalidad mínima de E_2 sea 1, es decir $I(E_1(0,n)):E_2(1,n)$, requiere que todo ejemplar de E_1 esté asociado con al menos uno de E_2 , pero no que todo ejemplar de E_2 esté vinculado con al menos uno de E_1 ¹³.

✱ En la figura 2.20 se muestran algunos ejemplares de la interrelación **Pertenece** entre DEPARTAMENTO y PROFESOR, en la que se ha supuesto que pueden existir departamentos que (por estar recién creados) no tienen ningún empleado y que todo empleado tiene que pertenecer siempre a un único departamento.

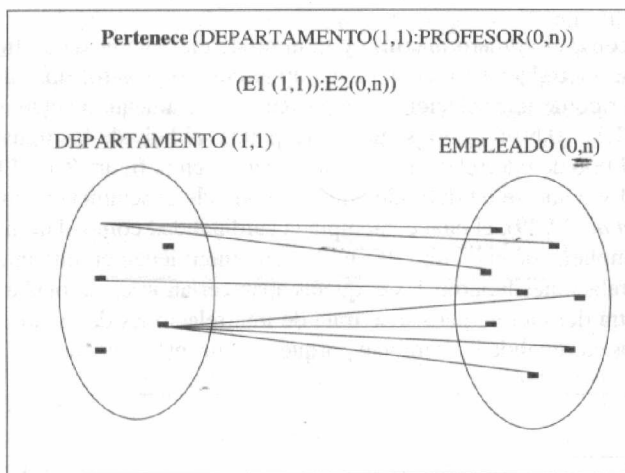


Figura 2.20. Representación de ejemplares de la interrelación **Pertenece**

¹² Otra de las razones que nos ha hecho inclinarnos por esta notación es que coincide con la utilizada para definir las cardinalidades entre atributo y entidad.

¹³ Téngase en cuenta que si las cardinalidades fuesen definidas de acuerdo con Tardieu, serían $I(E_1(1,n):E_2(0,n))$, diciéndose que es una aplicación total respecto a E_1 , la cual se representa con una doble línea en el arco que une E_1 con el rombo de la interrelación. Cuando la cardinalidad máxima de una interrelación es 1, la correspondencia es una función en sentido matemático, y se denomina *correspondencia funcional*.

Si bien las cardinalidades máximas coinciden con el tipo de correspondencia, la capacidad semántica es superior en las primeras, ya que las flechas con que se representa el tipo de correspondencia no permiten precisar, aunque se conozca, el número exacto de ejemplares vinculados en la interrelación.

4.3. Atributos de las interrelaciones

Cuando una interrelación 1:N tiene un atributo asociado (tal como aparece en la figura 2.21), es inmediata la demostración matemática (véase, por ejemplo, STOREY y GOLDSTEIN [1988]) de que el atributo puede llevarse a la entidad cuya cardinalidad máxima es N (en el ejemplo de la figura el atributo *Fecha_Imparte* podría llevarse a EDICIÓN), con independencia de los valores de las cardinalidades mínimas.

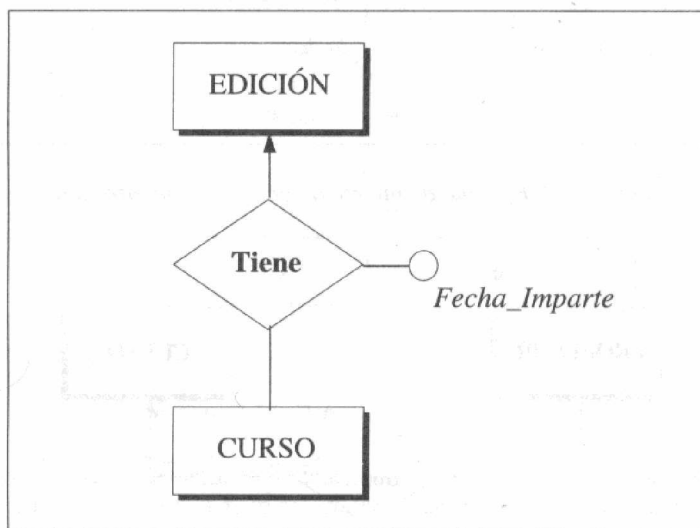


Figura 2.21. Interrelación 1:N con atributo

Semánticamente, sin embargo, puede ser, en ocasiones, de interés conservar el atributo dependiendo de la interrelación. Éste es el caso, por ejemplo, del esquema de la figura 2.22 donde tenemos el tipo de interrelación **Matrimonio** (1:1¹⁴) entre HOMBRE y MUJER, que tiene el atributo *fecha* (del matrimonio). Por ser la interrelación 1:1, para cada par (hombrex, mujery) existe una sola fecha válida de celebración del matrimonio, fecha que no es una propiedad de ninguno de los dos ejemplares, sino del hecho de la unión entre ellos, es decir, de la interrelación.

¹⁴ Se supone que la base de datos sólo recoge la información de los matrimonios actualmente vigentes.

Los atributos de las interrelaciones N:M, son propios de la misma y no de las entidades vinculadas por la interrelación; pueden incluso ser multivaluados como en el ejemplo de la figura 2.23 donde un profesor puede dar el mismo curso en varias fechas distintas, por lo que *Fecha* es un atributo multivaluado.

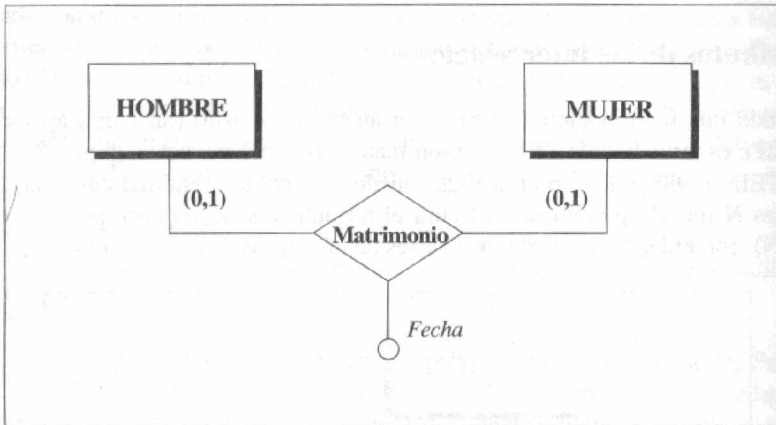


Figura 2.22. Ejemplo de interrelación 1:1 con un atributo

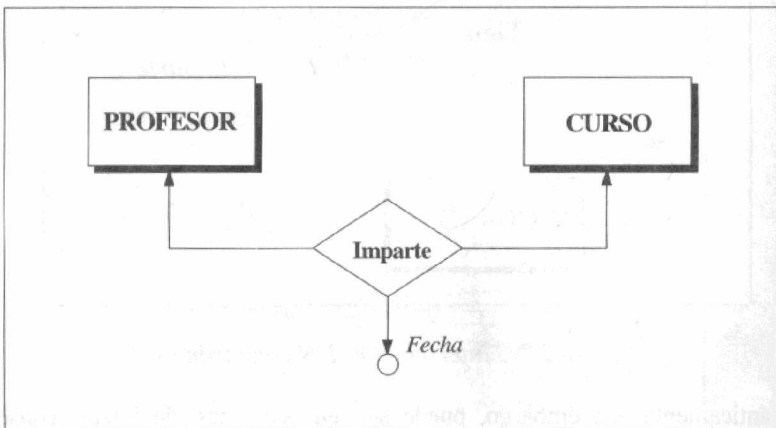


Figura 2.23. Ejemplo de interrelación N:M con un atributo multivaluado

4.4. Dependencia en existencia y en identificación

Como en el caso de los tipos de entidad, los tipos de interrelación se clasifican también en *regulares* y *débiles*, según estén asociando dos tipos de entidad regulares, o un tipo de entidad débil con un tipo de entidad (regular o débil), respectivamente. Es interesante distinguir, dentro del tipo de interrelación débil, la dependencia en *existencia*

y la dependencia en *identificación*. Se dice que hay dependencia en existencia cuando los ejemplares de un tipo de entidad (entidad débil) no pueden existir si desaparece el ejemplar del tipo de entidad regular del cual dependen. Se dice que existe dependencia en identificación cuando además de cumplirse la condición anterior, los ejemplares del tipo de entidad débil no se pueden identificar por sí mismos, es decir, mediante los propios atributos del tipo de entidad, y exigen añadir el identificador principal del tipo de entidad regular del cual dependen. Se ve claramente que una dependencia en identificación es siempre una dependencia en existencia (no ocurre lo contrario), y el tipo de interrelación es débil en ambos casos.

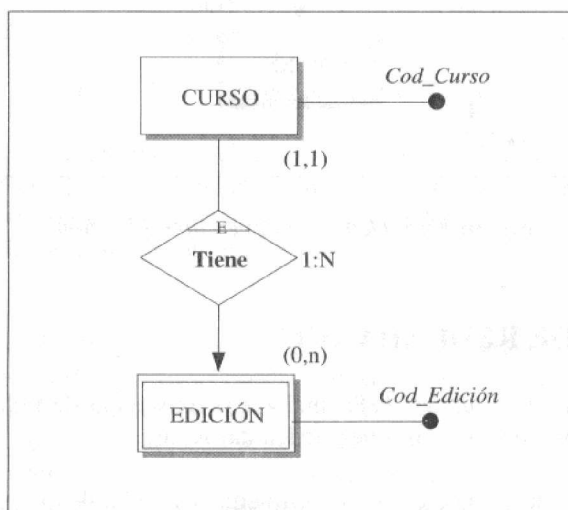


Figura 2.24. Dependencia en existencia

Si existe dependencia en identificación, el rombo que representa la interrelación va etiquetado con **ID**, y con una **E** (o sin etiqueta) en caso de que la dependencia sea en existencia. En la figura 2.24 se puede observar que los datos acerca de las ediciones de un curso sólo tendrán interés en tanto éste permanece en la base de datos, con lo que hay una dependencia en existencia. Sin embargo, cada edición tiene un identificador que lo distingue del resto independientemente del curso al que pertenezca. Por ejemplo, para el curso uno E1, E2, para el curso dos E3, E4, E5, etc.

En el supuesto de que las ediciones no tuvieran un identificador único, por ejemplo, E1, E2 fuesen ediciones del curso C1, E1, E2, E3 del curso C2, etc., entonces se dice que edición depende en identificación de curso. En la figura 2.25 se representa una dependencia en identificación donde se indica que el identificador de EDICIÓN (al que hemos llamado *Id_Edición*) se forma mediante el código de edición (*Cód_Edición*) más el identificador de la entidad de la cual depende EDICIÓN en la interrelación **Tiene**, es decir, *Cód_Curso*.

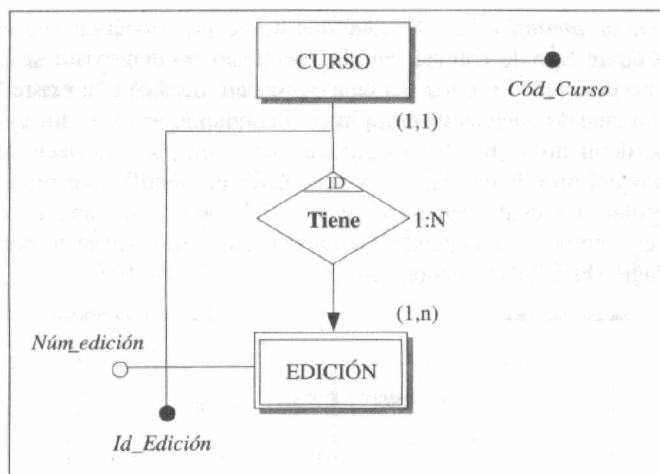


Figura 2.25. Dependencia en identificación

5. CONTROL DE REDUNDANCIA

Es preciso, en los esquemas E/R, analizar la existencia de redundancias, por lo problemas de inconsistencias a los que pueden dar lugar.

Decimos que un elemento de un esquema es redundante cuando puede ser eliminado sin pérdida de semántica.

Existen dos formas principales de redundancia, según el elemento del modelo E/R al que está asociada: redundancia en los atributos (atributos derivados) y redundancia en las interrelaciones (denominadas también por algunos autores interrelaciones derivadas).

5.1. Atributos derivados

Entendemos por atributos derivados (o calculados) aquellos que se obtienen a partir de otros ya existentes, por lo que, aunque son redundantes, no dan lugar a inconsistencias, siempre que en el esquema se indique su condición de derivados y la fórmula mediante la que han de ser calculados.

En la figura 2.26 tenemos el atributo *número de ediciones*, que puede ser calculado a partir de los ejemplares de edición mediante la interrelación **tiene**. Para indicarlo gráficamente utilizaremos la etiqueta **Di** en el atributo calificado como derivado, almacenando la regla de derivación en el diccionario de datos.

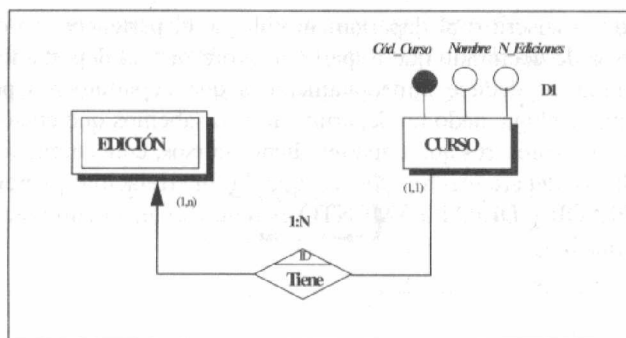


Figura 2.26. Ejemplo de atributo derivado

Incluir en el esquema conceptual atributos derivados, a pesar de que pueden ser generados a partir de otros ya existentes, tiene a veces interés por razones semánticas. Aunque también se podría hacer por motivos de eficiencia; sólo por esta causa no se deberían incluir dichos atributos en el esquema conceptual, sino en el lógico, o mejor aún, en el físico.

Un atributo derivado puede ser calculado en dos momentos distintos: bien en actualizaciones que pueden provocar cambios en su valor, bien cuando se recupera. En el primer caso, el atributo derivado se calcula y almacena (por lo que, por ejemplo, en el modelo de datos Codasyl se dice que es *real*); en el segundo no está almacenado y se calcula cuando se realiza una consulta (por lo que se dice que es *virtual*). El tomar una u otra decisión es propio del diseño físico, ya que se hace por motivos de eficiencia, y dependerá del número de actualizaciones frente al de recuperaciones. Tampoco hay que confundir un atributo derivado, cuyo valor no se introduce nunca sino que se calcula con las restricciones que comprueban la consistencia entre valores que están almacenados en la base de datos por haberlos introducido el usuario.

5.2. Interrelaciones redundantes

Se dice que una interrelación es redundante cuando su eliminación no implica pérdida de semántica porque existe la posibilidad de realizar la misma asociación de ejemplares por medio de otras interrelaciones.

Es condición necesaria, aunque no suficiente, para que una interrelación sea redundante que forme parte de un ciclo, por lo que hay que estudiar detenidamente los ciclos en el diagrama E/R.

En el ejemplo de la figura 2.27 se da un ciclo entre PROFESOR, CURSO y DEPARTAMENTO, por lo que en principio es posible que aparezca alguna interrelación redundante. Supongamos que un profesor sólo puede impartir cursos de

doctorado que estén adscritos al departamento al que él pertenece; en este caso, si se conocen los cursos de doctorado que imparte un profesor y el departamento al que está adscrito cada curso, se deduce inmediatamente a qué departamento pertenece dicho profesor; de forma análoga, dado un departamento, si sabemos qué cursos de doctorado tiene adscritos y los profesores que imparten dichos cursos, conoceremos qué profesores pertenecen a dicho departamento, por lo que la interrelación **pertenece** entre las entidades **PROFESOR** y **DEPARTAMENTO** es redundante, su eliminación no produce pérdida de información.

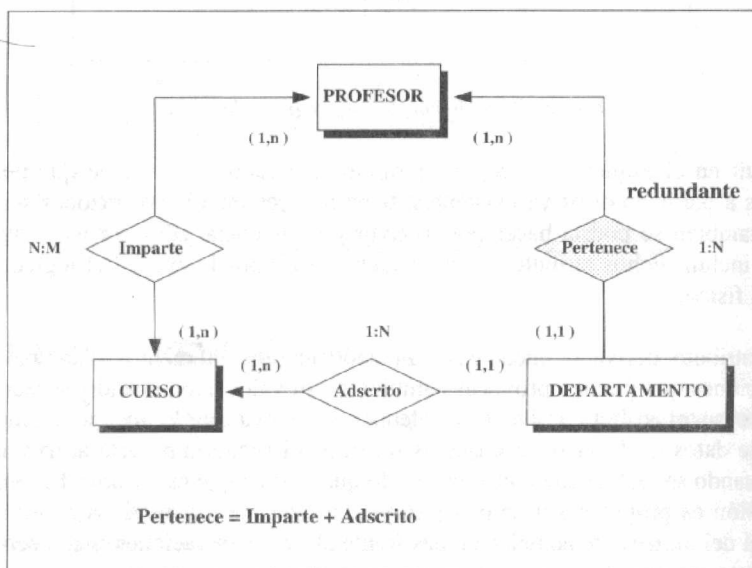


Figura 2.27. Ciclo en el que aparece una interrelación redundante

En la figura 2.28, a pesar de que también existe un ciclo, no hay ninguna interrelación redundante. En este ejemplo la semántica es distinta y un departamento puede no tener adscritos cursos de doctorado; además un mismo curso puede estar adscrito a distintos departamentos y puede haber profesores que no impartan ningún curso. La interrelación **pertenece** no puede deducirse en este caso de las otras dos, ya que aunque sepamos los cursos que ha impartido un profesor y los departamentos a los que están adscritos dichos cursos, no podemos saber a qué departamento en concreto pertenece dicho profesor; tampoco se tiene esta información para los profesores que no imparten ningún curso. La interrelación **imparte** tampoco es redundante, ya que un curso de doctorado puede ser impartido por diversos departamentos a cada uno de los cuales pertenecen varios profesores, por lo que no se puede saber qué profesor en concreto imparte un determinado curso. Por último, la interrelación **adscrito** tampoco es redundante, ya que un curso impartido por un profesor no tiene por qué estar necesariamente adscrito al departamento al que pertenece dicho profesor: hay

departamentos que no tienen cursos adscritos y los profesores de estos departamentos pueden colaborar en cursos adscritos a otros departamentos distintos del suyo.

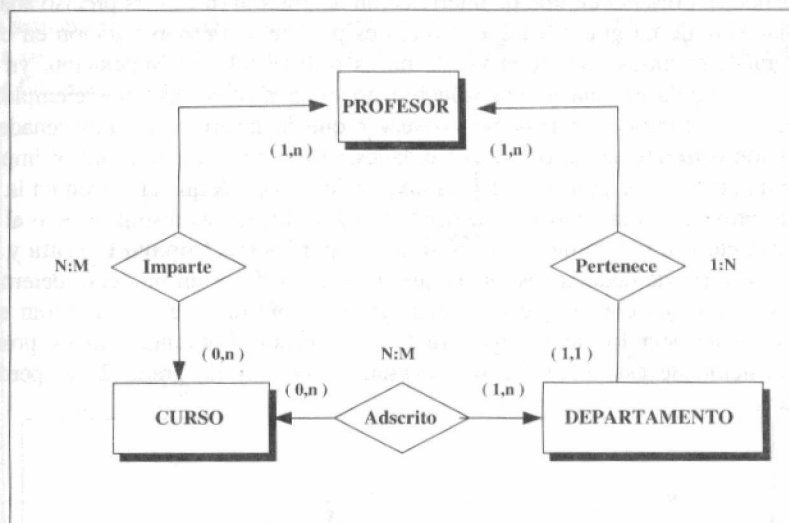


Figura 2.28. Ciclo en el que no aparece una interrelación redundante

Existen otros casos en los que la interrelación, a pesar de poder ser deducida a partir de otras presentes en el esquema, no se puede eliminar porque posee atributos.

Se puede decir, como norma general, que la existencia de un ciclo no implica la existencia de interrelaciones redundantes. Deben estudiarse con mucho detenimiento las cardinalidades mínimas de las entidades, así como la semántica que aportan las interrelaciones, para poder afirmar con seguridad que existen interrelaciones redundantes. Habrá que analizar si al eliminar una interrelación es *siempre* posible el paso, tanto en un sentido como en el inverso, entre las dos entidades unidas por la interrelación que se considera redundante, y habrá que comprobar también que no se pierdan atributos.

En resumen, para que una interrelación pueda ser eliminada por redundante se tiene que cumplir:

- Que exista un ciclo
- Que las interrelaciones que componen el ciclo sean equivalentes semánticamente
- Que se puedan asociar los ejemplares de las dos entidades que estaban interrelacionadas, aún habiéndose eliminado la interrelación
- Que la interrelación o bien no tenga atributos o bien éstos puedan ser transferidos a otra a fin de no perder su semántica.

6. INTERRELACIONES DE GRADO SUPERIOR A 2

Cuando se presenta un tipo de interrelación de grado n ($n > 2$), es preciso analizar si es propiamente de tal grado, ya que a veces es posible su descomposición en otras de menor grado; mientras que, otras veces, no es posible tal descomposición, ya que la semántica recogida en una y otra solución no es la misma. Así, por ejemplo, en el esquema de la figura 2.29 podemos observar que la información almacenada en la interrelación **Imparte**, que asocia tres entidades, se refiere a que un profesor imparte un tema en un curso (se supone que las cardinalidades¹⁵ son las que aparecen en la figura, donde un profesor en un cierto curso puede tratar varios temas distintos, pero al menos tratará uno, etc.); si sustituimos esta interrelación por las tres **Imparte1**, **Trata** y **Entra**, de ellas no se puede deducir los temas que trata un profesor en un curso determinado, aunque sepamos los cursos que ha impartido ese profesor, qué temas entran en esos cursos y cuáles son los temas que trata ese profesor. Por tanto, no es posible la descomposición de esta interrelación de grado 3 en tres de grado 2 sin pérdida de semántica.

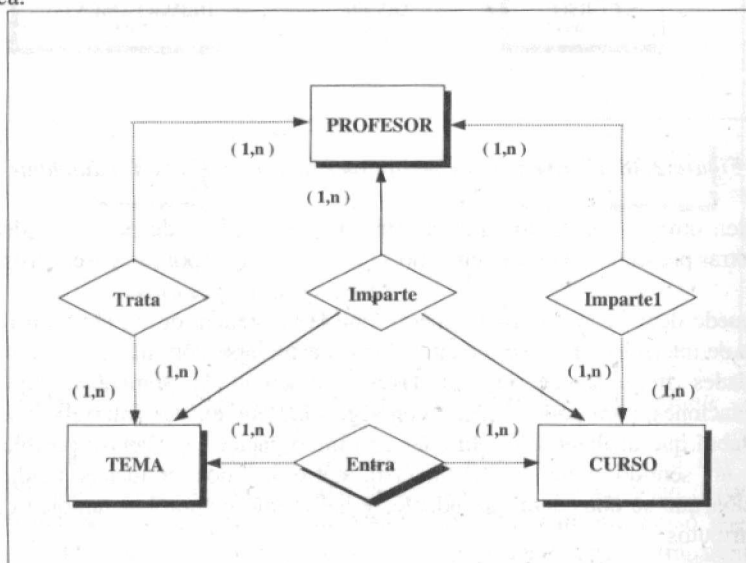


Figura 2.29. Ejemplo de un tipo de interrelación de grado 3 que no puede ser descompuesta sin pérdida de semántica

¹⁵ Tal como hemos definido las cardinalidades, en una interrelación de grado 3 la cardinalidad de una de las entidades (E_1) con respecto a las otras dos (E_2 y E_3) es el número mínimo y máximo de ejemplares de E_1 que están vinculados con uno de E_2 y de E_3 ya vinculados en la interrelación. Obsérvese que los valores de las cardinalidades así definidas pueden ser distintos de los de las cardinalidades tal como fueron definidas por Tardieu y aparecen en muchos libros.

En la figura 2.30, sin embargo, se muestra la interrelación **Imparte** entre PROFESOR, CURSO y ESTUDIANTE que sí puede ser descompuesta sin perder semántica en las interrelaciones **Imparte1**, **Da_clase** y **Asiste**, ya que éstas aportan la misma semántica que la interrelación de grado tres¹⁶. Cuando un tipo de interrelación de grado n ($n > 2$) puede ser sustituido por otros de grado menor, sin pérdida de semántica, se debe llevar a cabo tal sustitución¹⁷.

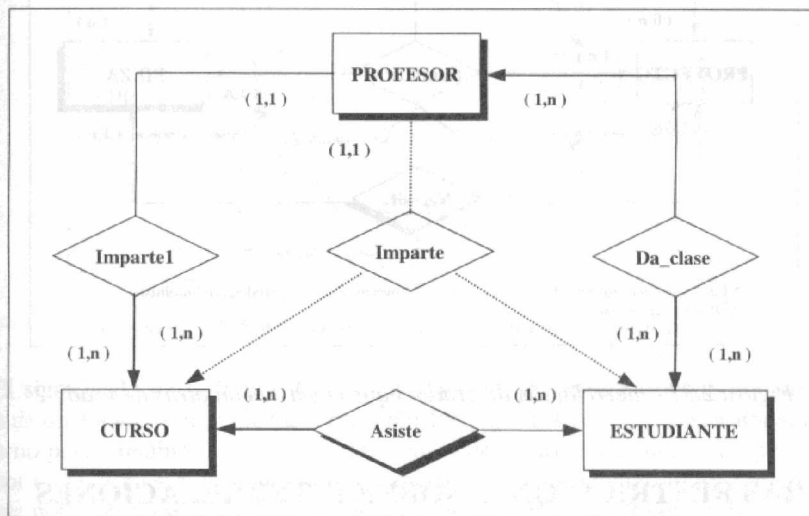


Figura 2.30. Ejemplo de descomposición, sin pérdida de semántica, de un tipo de interrelación de grado 3

La existencia de una interrelación de grado superior a 2 no es incompatible con la existencia de interrelaciones de menor grado en las que participen los mismos tipos de entidad. Por ejemplo, en la figura 2.31 la interrelación de grado 3 **Suministra** coexiste con las tres interrelaciones de grado 2 (**Puede suministrar**, **Interviene** y **Necesita**), ya que éstas recogen las piezas que puede suministrar un proveedor o para los proyectos que puede suministrar, etc., mientras que la de grado 3 representa las piezas que, de hecho, están siendo suministradas para un cierto proyecto por un determinado proveedor¹⁸; por tanto, la semántica de la interrelación ternaria es distinta de la de las interrelaciones binarias y el usuario podría necesitar que se mantuvieran tres interrelaciones (**Interviene** sí es redundante con respecto a **Suministra**).

¹⁶ Incluso basta con dos interrelaciones, **Imparte1** y **Asiste**, para reflejar toda la semántica de la interrelación original, ya que la interrelación **Da_clase** es redundante.

¹⁷ En la posibilidad de descomposición de tipos de interrelación de grado superior a dos en otros de menor grado influyen las cardinalidades.

¹⁸ Obsérvese que haciendo otros supuestos semánticos, las cardinalidades podrían cambiar y también podría resultar redundante alguna de las interrelaciones.

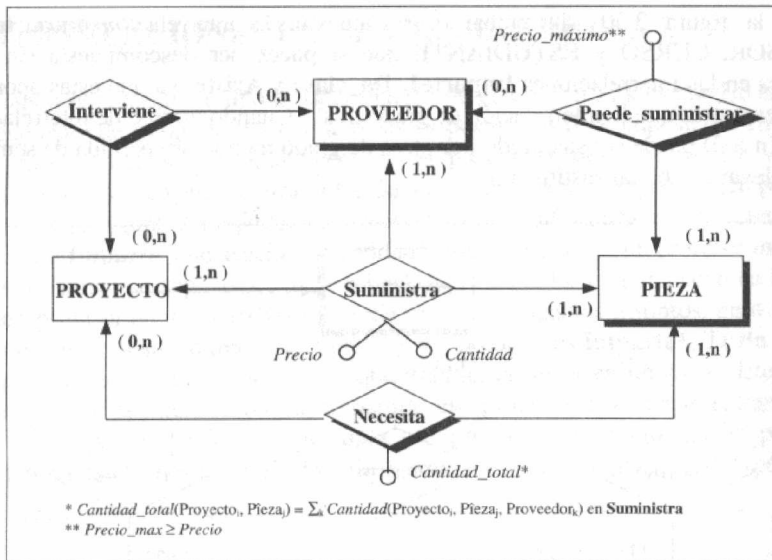


Figura 2.31. Interrelación de grado 3 que coexiste con otras de grado 2

7. OTRAS RESTRICCIONES SOBRE INTERRELACIONES

Existen, además de las vistas hasta ahora, otras restricciones que afectan a los tipos de interrelación y a sus ejemplares, como son: restricción de exclusividad, restricción de exclusión, restricción de inclusividad y restricción de inclusión. Se trata de extensiones del modelo E/R que no es habitual recoger en conjunto, ni tampoco lo es diferenciar entre exclusión y exclusividad o entre inclusión e inclusividad. Así, por ejemplo, en DE MIGUEL y PIATTINI (1992) se hablaba de interrelaciones exclusivas; en Merise, ROCHFELD (1992), se introduce el concepto de exclusión pero no el de exclusividad; en OMT, RUMBAUGH *et al.* (1991) o en UML, BOOCH *et al.* (1997) tan sólo se considera la restricción de inclusión. Un estudio más profundo de las restricciones sobre tipos de interrelación puede encontrarse en MARCOS (1997).

7.1. Restricción de Exclusividad

Decimos que dos (o más) tipos de interrelación tienen una restricción de exclusividad con respecto a un tipo de entidad que participa en ambas interrelaciones cuando cada ejemplar de dicho tipo de entidad sólo puede pertenecer a uno de los tipos de la interrelación, pero en el momento en que pertenezca a uno ya no podrá formar parte del otro. Por ejemplo, si suponemos que un profesor puede impartir cursos de doctorado o recibirlos, pero no ambas cosas, tendríamos una interrelación **Imparte** y

otra **Recibe**, entre PROFESOR y CURSO, con una restricción de exclusividad entre sí. En la figura 2.32 se muestra la representación de la exclusividad. El arco señala las interrelaciones que son exclusivas;

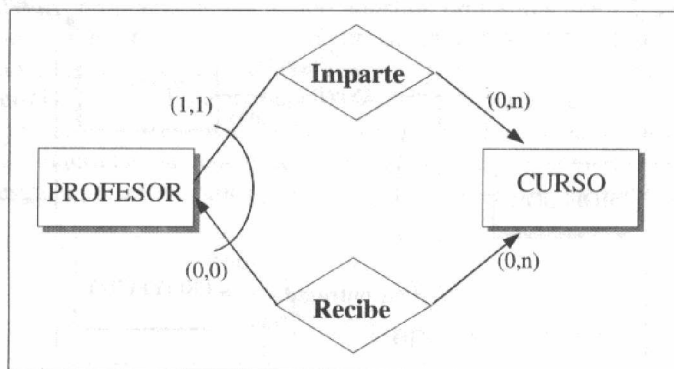


Figura 2.32. Ejemplo de tipo de interrelación “exclusiva”

El significado de la figura 2.32 es el siguiente: un profesor puede impartir o no cursos de doctorado (0,n), y puede o no recibirlos (0,n), pero si un profesor imparte estos cursos no puede recibirlos y viceversa. Un curso de doctorado es impartido por un solo profesor (1,1), pero a él pueden asistir varios profesores o ninguno (0,n). Sin embargo, con esta notación no se representa la cardinalidad de PROFESOR con respecto a ambas interrelaciones, o dicho de otro modo, no sabemos si es obligatorio que un profesor tenga que impartir o bien recibir un curso. En la figura 2.33 se muestra otra notación para las interrelaciones exclusivas en la que, además de la cardinalidad de PROFESOR con respecto a **Imparte** y **Recibe**, por separado, se muestra la cardinalidad de PROFESOR con respecto a ambas interrelaciones.

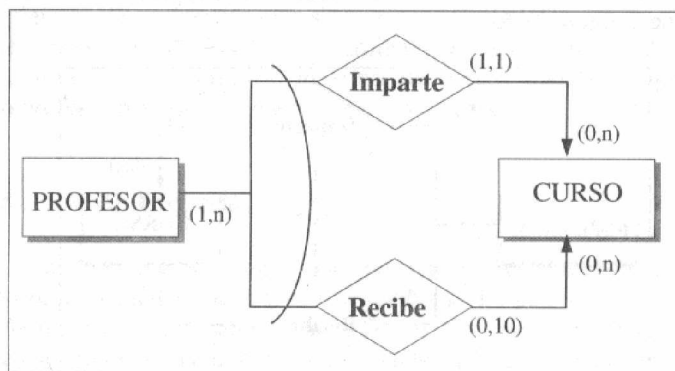


Figura 2.33. Ejemplo de tipo de interrelación “exclusiva” con otra notación que permite captar más semántica

No es obligatorio que las interrelaciones exclusivas lo sean respecto al mismo tipo de entidad (en este caso **CURSO**), sino que podrían serlo respecto a distintos tipos. Véase, por ejemplo, figura 2.34, donde si un profesor percibe una beca no puede estar contratado en un proyecto.

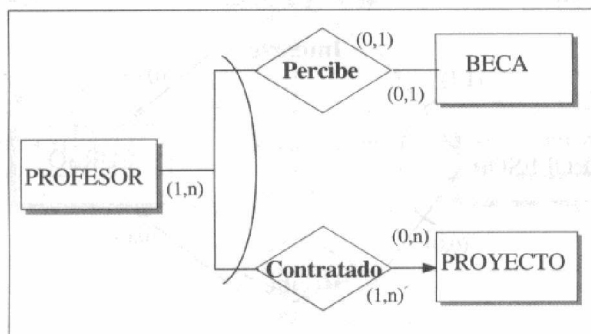


Figura 2.34. Ejemplo de interrelaciones exclusivas de un tipo de entidad respecto a dos

7.2. Restricción de Exclusión

La restricción de exclusividad en el ejemplo anterior indicaba que un profesor podía impartir o recibir cursos, pero no ambas cosas; si el profesor no es doctor podrá recibir cursos de doctorado y en caso contrario impartirlos. Supongamos ahora que se permite a un profesor ya doctor matricularse en cursos aunque él, a su vez, esté impartiendo otros cursos. En este caso la restricción que debemos imponer es que un profesor no esté impartiendo y recibiendo el mismo curso. Es decir, que todo ejemplar de profesor que esté unido a un ejemplar de curso mediante la interrelación **imparte**, no podrá estar unido al mismo ejemplar de curso mediante la interrelación **recibe**. En este caso decimos que existe una restricción de exclusión y se representa tal y como aparece en el ejemplo de la figura 2.35.

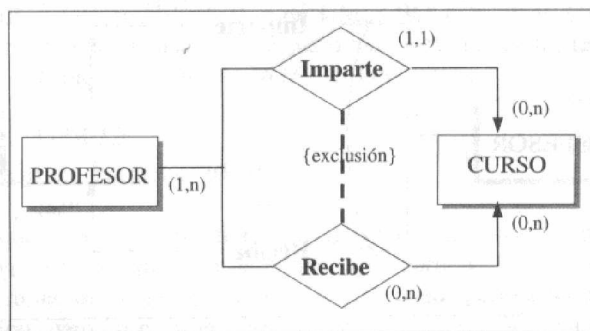


Figura 2.35. Ejemplo de tipo de interrelación con restricción de exclusión

7.3. Restricción de Inclusividad

Supongamos ahora que se desea imponer la restricción de que sólo pueden impartir clases en nuestro programa de doctorado aquellos profesores que hayan realizado al menos un curso dentro de este mismo programa, aunque no tiene por qué ser el mismo que él imparte. Aplicamos entonces una restricción de inclusividad entre dos (o mas) tipos de interrelación con respecto a uno de los tipos de entidad que participa en ambas interrelaciones, por la cual toda ejemplar de dicho tipo de entidad que participa en uno de los tipos de interrelación tiene necesariamente que participar en la otra. En la figura 2.36 se muestra la notación gráfica propuesta para este tipo de interrelación.

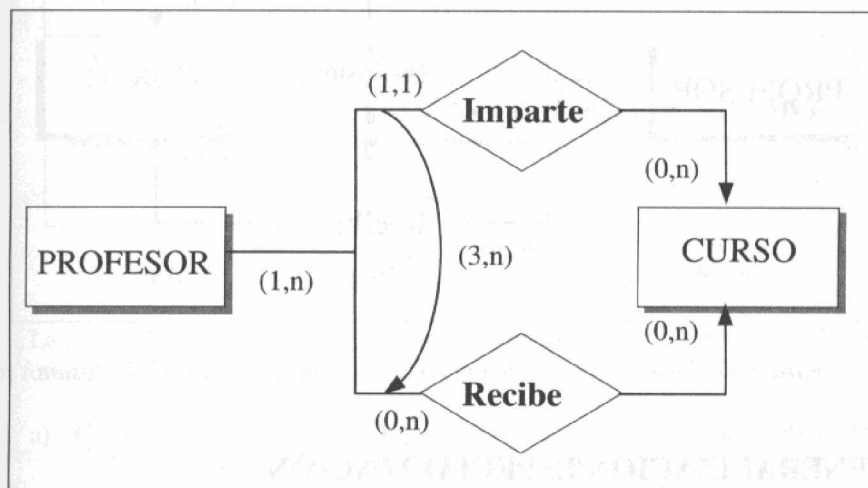


Figura 2.36. Ejemplo de tipo de interrelación con restricción de inclusividad

En este ejemplo se representa que si un profesor participa en Imparte tiene que participar necesariamente en Recibe. La cardinalidad sobre la flecha de inclusividad, (3,n), indica el número mínimo y máximo de cursos que tiene que recibir un determinado profesor para que se le permita impartir cursos.

7.4. Restricción de Inclusión

A veces es preciso imponer una restricción más fuerte: si un profesor imparte un curso es porque previamente ha tenido que recibir dicho curso. Aplicamos pues una restricción de inclusión, representada en la figura 2.34, por la cual todo ejemplar de profesor que esté unido a un ejemplar de curso mediante la interrelación imparte, tiene necesariamente que estar unido al mismo ejemplar de curso mediante la interrelación recibe.

Si se considera la dimensión temporal se pueden tener casos más complejos de modelado, como por ejemplo que todo profesor que imparte un curso tiene que haberlo recibido antes (restricción de inclusión con el histórico de recibe) pero no puede estar recibéndolo a la vez que lo imparte (restricción de exclusión con el actual de recibe).

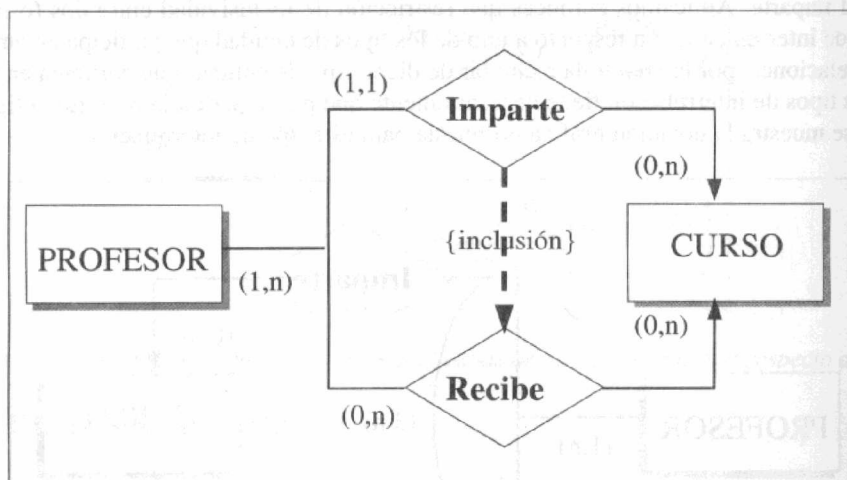


Figura 2.37. Ejemplo de tipo de interrelación con restricción de inclusión

8. GENERALIZACIÓN/ESPECIALIZACIÓN

En el modelo E/R básico propuesto por CHEN (1976) no se encontraba este tipo de abstracción que fue introducido en posteriores extensiones del modelo. Tiene su origen en el campo de la inteligencia artificial, introducido por QUILLIAN (1968) en las *redes semánticas*, habiendo sido adoptado en varios modelos de datos debido a la capacidad semántica que ofrece para la representación del mundo real. La jerarquía de generalización/especialización, en el modelo E/R, se considera como un caso especial de interrelación entre varios tipos de entidad (*subtipos*) y un tipo más general (*supertipo*) cuyas características son comunes a todos los subtipos. La interrelación que se establece entre los subtipos y el supertipo corresponde a la noción de “*es_un*”¹⁹ o más precisamente “*es_un_tipo_de*”.

Aunque existen distintas convenciones para representar estas jerarquías de generalización/especialización, nosotros utilizamos un triángulo cuya base es paralela al rectángulo que representa la entidad del supertipo al cuál está conectado; triángulo que también se une a los subtipos, tal como se muestra en la figura 2.38.

¹⁹ En inglés IS_A

Esta clase de interrelación tiene la característica de que todo ejemplar de un subtipo es también un ejemplar del supertipo, aunque no sucede lo contrario, con lo que las cardinalidades serán siempre (1,1) en el supertipo y (0,1) en los subtipos.

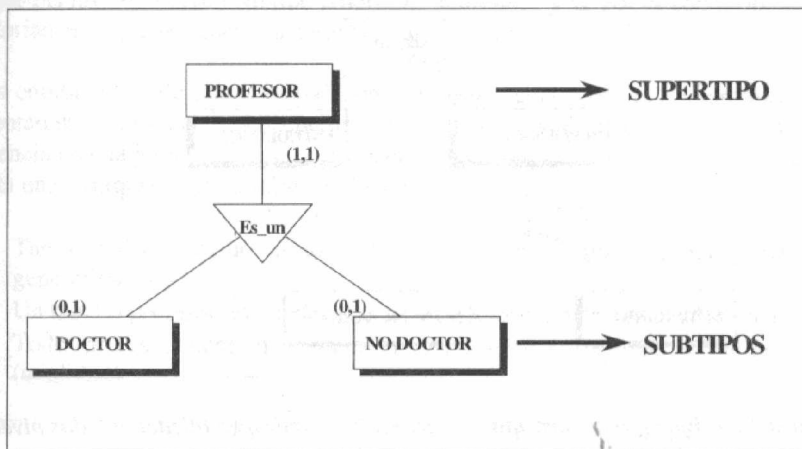


Figura 2.38. Ejemplo de jerarquía de supertipo/subtipos

La aparición de estas jerarquías en el modelado de bases de datos puede surgir de dos formas distintas:

- Generalización.** Se observa que dos o más tipos de entidad comparten varios atributos y/o tipos de interrelación, de donde se deduce la existencia de un tipo de entidad de nivel superior (supertipo) que contiene los atributos y los tipos de interrelación comunes a todos los subtipos.
- Especialización.** Se observa que un tipo de entidad tiene ciertos atributos y/o tipos de interrelación que tienen sentido para unos ejemplares pero no para otros, por lo que es conveniente definir uno o varios subtipos que contengan estos atributos y/o tipos de interrelación específicos, dejando en el supertipo los que son comunes.

Por tanto, si nos movemos de los subtipos hacia el supertipo, se trata de una generalización; mientras que si primero identificamos el supertipo y, a partir de él, llegamos a los subtipos, se trata de una especialización.

Puede ocurrir que se formen, por generalización y/o especialización, jerarquías a más de un nivel donde un subtipo es, a su vez, supertipo de otros, como ocurre en la figura 2.39, donde se puede observar una jerarquía a dos niveles donde uno de ellos se ha obtenido por generalización de profesor y estudiante en persona, y el otro nivel por especialización de profesor en numerario y no numerario.

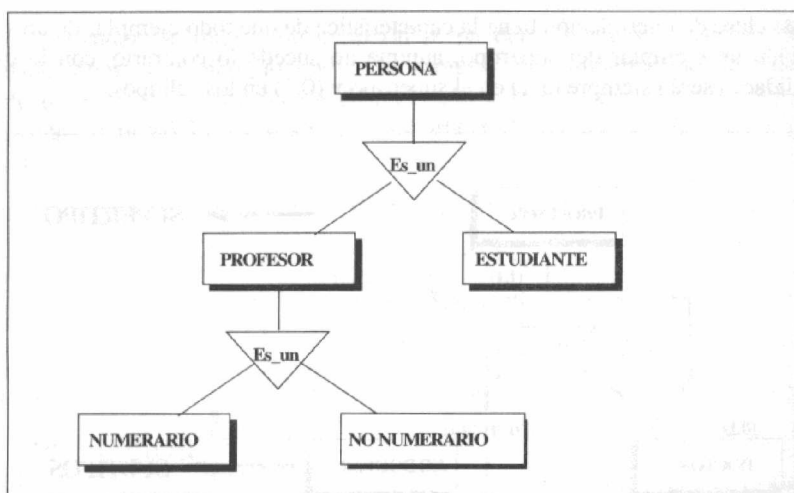


Figura 2.39. Ejemplo de jerarquía de generalización/especialización a dos niveles

Otra característica muy importante de esta clase de interrelaciones es la herencia, ya que, en principio todo atributo del supertipo pasa a ser un atributo de los subtipos; por ejemplo, en la jerarquía de la figura 2.38 tanto los doctores como los no doctores *son* (o *son tipos de*) profesores, por lo que heredarán todos los atributos de PROFESOR (Código, Nombre, DNI, Dirección, etc.). Esta característica la diferencia de la clasificación, donde los subtipos son ejemplares por lo que al heredar los atributos del supertipo lo hacen tomando valores para cada uno de los atributos heredados, mientras que en la generalización propiamente dicha se heredan los atributos, pero sin sus valores.

En este tipo de abstracción los atributos comunes a todos los subtipos (incluidos los identificadores) se asignan al supertipo, mientras que los atributos específicos se asocian al subtipo al cual pertenecen. Del mismo modo, las interrelaciones que afectan a todos los subtipos se asocian al supertipo, dejándose para los subtipos las interrelaciones específicas en las que sólo participa el correspondiente subtipo.

La división en subtipos (*especialización*) puede venir determinada por una condición predefinida (por ejemplo, en función de los valores de un atributo) en cuyo caso se representará la condición (o el atributo *discriminante*) asociada al triángulo que representa la interrelación. Si no interesa considerar ninguna condición predefinida, deberá ser el usuario, en el momento de insertar un ejemplar en la base de datos, quién especifique a cuál de los subtipos pertenece.

La abstracción de generalización/especialización tiene algunas restricciones semánticas de las que nos ocuparemos a continuación. Atendiendo a si los subtipos se solapan o son disjuntos, y a si la unión de los subtipos recubre o no al supertipo, se

pueden distinguir cuatro clases de generalización. Si un mismo ejemplar del supertipo puede pertenecer a más de un subtipo habrá solapamiento, y si sólo puede pertenecer a uno de los subtipos existirá *exclusividad*; por otro lado, si todo ejemplar del supertipo tiene que pertenecer a algún subtipo tendremos *totalidad*, y si, por el contrario, no tiene obligatoriamente que pertenecer a algún subtipo habrá *parcialidad*.

La combinación de estas posibilidades da lugar a cuatro tipos de jerarquías, donde representaremos por un arco el hecho de que los subtipos sean disjuntos y con un círculo la presencia de una jerarquía total, como puede observarse en la figura 2.40, en la cual se presenta una jerarquía total de subtipos disjuntos, ya que:

- Tanto un doctor como un no doctor son profesores (por tener una jerarquía de generalización)
- Un mismo profesor no puede ser a la vez doctor y no doctor (exclusividad)
- Todo profesor tiene que ser obligatoriamente un doctor o un no doctor (totalidad)

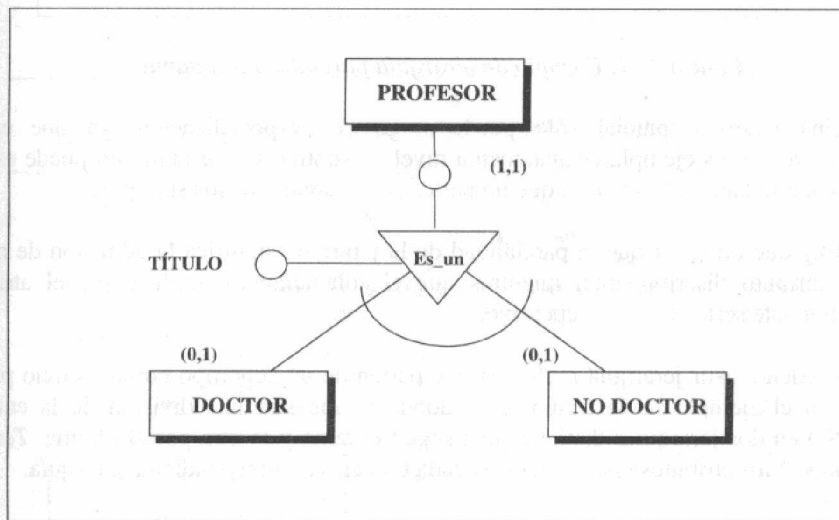


Figura 2.40. Ejemplo de jerarquía total sin solapamiento

En la figura 2.41 se puede observar cómo el supertipo DOCUMENTO y los subtipos LIBRO y ARTÍCULO forman una jerarquía disjunta y parcial, que se traduciría en lo siguiente:

- Tanto un artículo como un libro son documentos
- Un mismo documento no puede ser a la vez un artículo y un libro (exclusividad)
- Un documento puede no ser ni un artículo ni un libro (parcialidad)

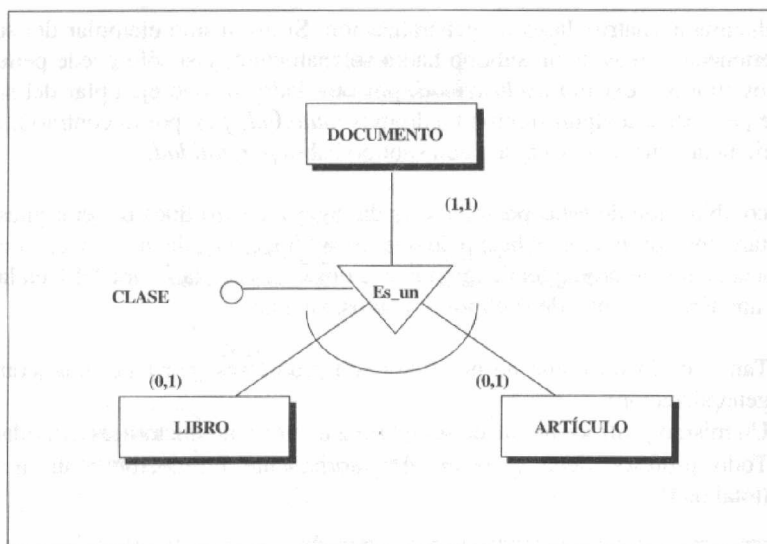


Figura 2.41. Ejemplo de jerarquía parcial sin solapamiento

Una jerarquía parcial sólo puede surgir por especialización, ya que en la generalización los ejemplares aparecen a nivel de subtipo y, por tanto, no puede existir ningún ejemplar en el supertipo que no pertenezca a alguno de los subtipos.

Hay que observar que la parcialidad de la jerarquía significa la admisión de nulos en el atributo discriminante, mientras que el solapamiento implica que el atributo discriminante sería un grupo repetitivo.

Pueden existir jerarquías múltiples que parten de un supertipo común, como puede verse en el ejemplo de la figura 2.42, donde se muestra una división de la entidad CURSO en dos jerarquías distintas, una según el tema y la otra por el idioma; *Tema* e *Idioma* son los atributos discriminantes, cada uno en su correspondiente jerarquía.

Una forma alternativa, o más bien complementaria, de representar una abstracción de generalización/especialización ha sido propuesta en WAGNER (1988), y consiste en *tablas jerárquicas*, que permiten representar la herencia con todas sus características, de manera clara y concisa, sobre todo en el caso de existir varias jerarquías que parten de un mismo supertipo.

Estas tablas, de las que se muestra un ejemplo en la figura 2.43 (relativo a la jerarquía de la figura 2.42), representan mediante "1" las combinaciones posibles. Además, en la parte inferior de la tabla se refleja la jerarquía a la que pertenecen las entidades (cuando se trate de una raíz estará vacía); el tipo de jerarquía (D-disjunta, S-solapada, P-parcial, T-total), aparece en la entrada que tiene como etiqueta "definida

como”; el atributo discriminante sobre el que se define la jerarquía y la entidad de la que el subtipo hereda tienen también sus correspondientes entradas.

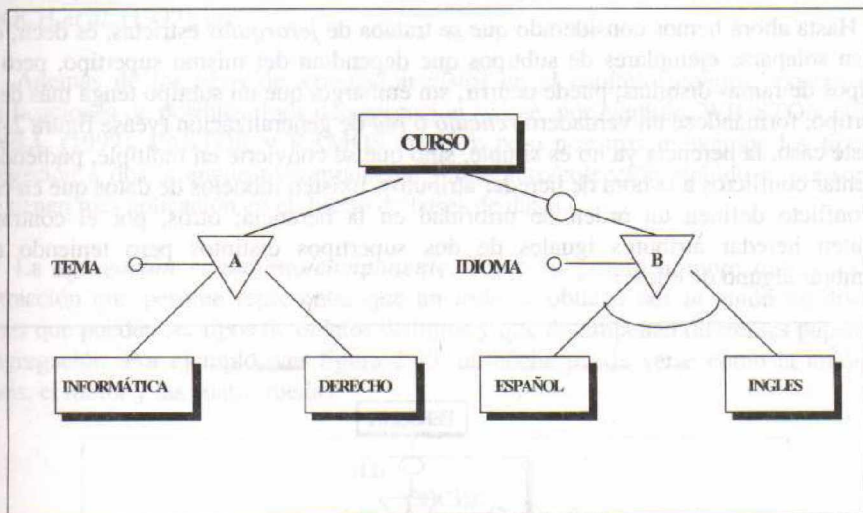


Figura 2.42. Ejemplo de jerarquías múltiples

	TIPOS DE ENTIDAD				
	CURSO	INFORMÁTICA	DERECHO	INGLÉS	ESPAÑOL
combinaciones (a)	1	1	0	1	0
permitidas (b)	1	1	0	0	1
(c)	1	0	1	0	1
(d)	1	1	1	0	1
(e)	1	0	0	1	1
jerarquía	-	A	A	B	B
definida como	RAÍZ	SP	SP	DT	DT
definida sobre (atributo discriminante)	-	TEMA	TEMA	IDIOMA	IDIOMA
hereda de	-	CURSO	CURSO	CURSO	CURSO

Figura 2.43. Tabla de representación de jerarquías de generalización

Así, en el ejemplo se especifica que sólo pueden darse cinco posibilidades: cursos de informática en inglés o en español, cursos de derecho en español, cursos de informática y derecho sólo en español y cursos en inglés que no son ni de informática ni

de derecho; cualquiera de las otras posibilidades no se admite en nuestro universo del discurso.

Hasta ahora hemos considerado que se trataba de *jerarquías* estrictas, es decir, que podían solaparse ejemplares de subtipos que dependían del mismo supertipo, pero no subtipos de ramas distintas; puede ocurrir, sin embargo, que un subtipo tenga más de un supertipo, formándose un verdadero *retículo* o *red* de generalización (véase figura 2.44). En este caso, la herencia ya no es simple, sino que se convierte en múltiple, pudiéndose presentar conflictos a la hora de heredar atributos. Existen modelos de datos que en caso de conflicto definen un orden de prioridad en la herencia; otros, por el contrario, permiten heredar atributos iguales de dos supertipos distintos pero teniendo que renombrar alguno de ellos.

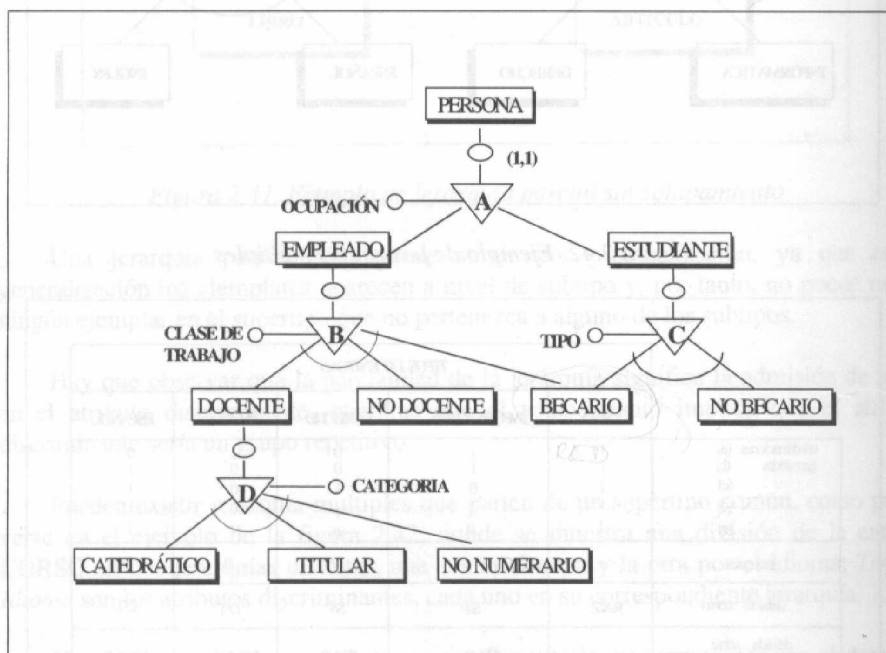


Figura 2.44. Ejemplo de red de generalización

9. AGREGACIÓN

La agregación, también llamada por algunos autores meronimia, es una abstracción (ya expuesta en el capítulo anterior) que permite representar tipos de entidad compuestos que se obtienen por unión de otros más simples. Al tipo compuesto nos referimos como

el *todo*, mientras que los componentes son las *partes*. Esta extensión del modelo E/R no aparece en su primera versión del mismo, pero se recoge posteriormente, en especial en todas las propuestas relativas al modelado de objetos, RUMBAUGH *et al.* (1991), BOOCH *et al.* (1997), etc.

Además de los tipos de agregación vistos en el capítulo anterior, existen otras clasificaciones de posibles tipos de agregación (véase, por ejemplo, WINSTON (1987), ODELL (1997), PASTOR y RAMOS (1995)); pero nosotros reducimos los tipos de agregación a dos: compuesto/componente y miembro/colección, debido a que son los que tienen más aplicación en el diseño de bases de datos.

La **agregación compuesto/componente**, como su propio nombre indica, es una abstracción que permite representar que un *todo* se obtiene por la unión de diversas *partes* que pueden ser tipos de objetos distintos y que desempeñan diferentes papeles en la agregación. Por ejemplo, ver figura 2.45, un coche puede verse como la unión del chasis, el motor y las cuatro ruedas.

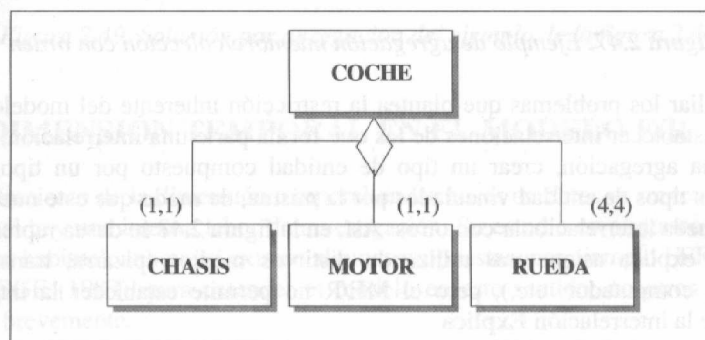


Figura 2.45. Ejemplo de agregación compuesto/componente

La agregación miembro/colección es la abstracción que permite representar un *todo* como una colección de partes, donde todas las partes son de un mismo tipo y desempeñan el mismo papel. Por ejemplo, en la figura 2.46 se puede observar cómo un bosque es un todo formado por la agregación de árboles; cada árbol es una parte, pero todos ellos son de un mismo tipo y desempeñan el mismo papel.

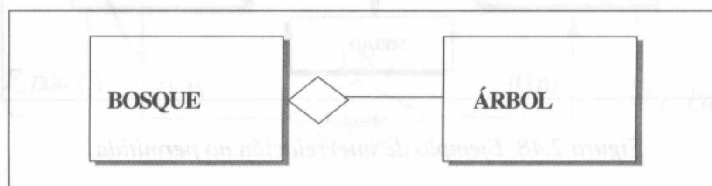


Figura 2.46. Ejemplo de agregación miembro/colección

En la agregación miembro/colección a veces se desea establecer un orden²⁰ entre las partes. Por ejemplo, una flota está compuesta por barcos pero, a diferencia de lo que ocurre con el bosque, en la flota cada barco tiene un determinado orden. Esto se representa mediante una restricción de orden, tal y como se puede observar en la figura 2.47, donde los barcos se ordenan, dentro de la flota, según el valor del atributo *Núm_barco*. Esta restricción se puede recoger, igualmente, en los actuales modelos de objetos. Sin embargo, en el diseño lógico en el modelo relacional, esta restricción no se puede recoger directamente.

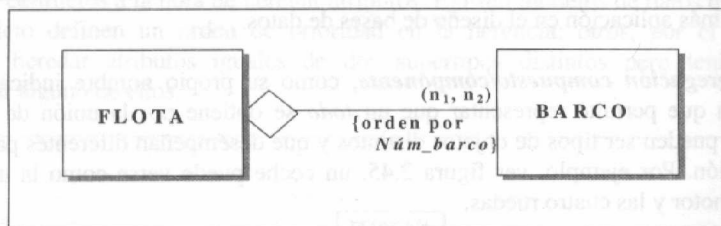


Figura 2.47. Ejemplo de agregación miembro/colección con orden

Para paliar los problemas que plantea la restricción inherente del modelo E/R que no permite establecer interrelaciones de las que forma parte una interrelación, se puede, mediante una agregación, crear un tipo de entidad compuesto por un tipo de interrelación y los tipos de entidad vinculados por la misma, de modo que este nuevo tipo de entidad se pueda interrelacionar con otros. Así, en la figura 2.48 se desea representar que un profesor explica asignaturas utilizando distintos medios (pizarra, transparencias, diapositivas, computador, etc.), pero el ME/R no permite establecer la interrelación **Utiliza** sobre la interrelación **Explica**.

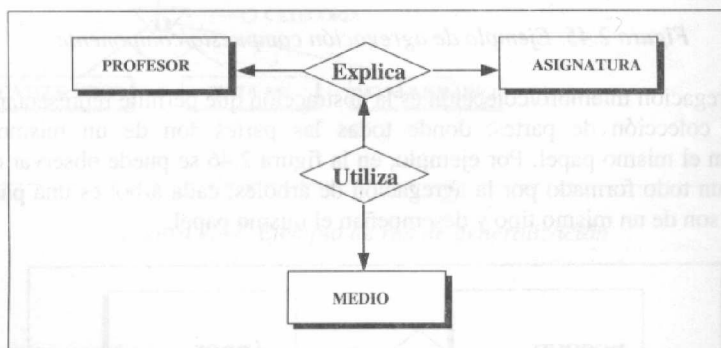


Figura 2.48. Ejemplo de interrelación no permitida

²⁰ Esta restricción de orden también se podría establecer entre ejemplares de entidades asociados mediante interrelaciones.

Una solución a este problema aparece en la figura 2.49, en la cual se crea un tipo de entidad “EXPLICACIÓN” por agregación de PROFESOR *Explica* ASIGNATURA.

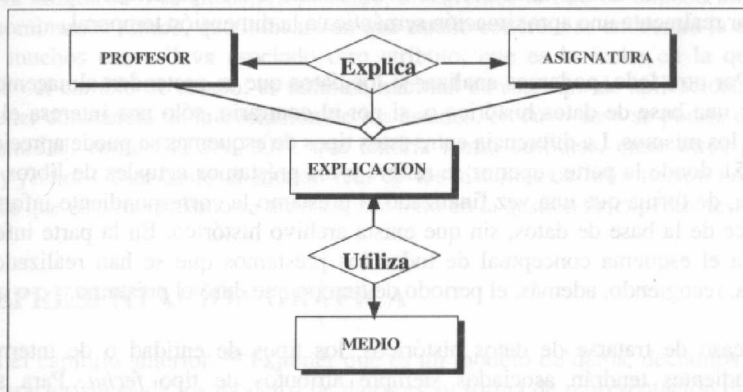


Figura 2.49. Solución por agregación del ejemplo de la figura 2.48

10. LA DIMENSIÓN TEMPORAL EN EL MODELO E/R

El tratamiento de la dimensión temporal en las bases de datos es un tema complejo sobre el cual hay una intensa labor de investigación. Su estudio en el marco del modelo E/R es poco habitual, aunque sí existen algunas propuestas, por ejemplo FERG (1985) y KLOPROGGE (1983), para extender el modelo en este sentido; nosotros lo vamos a tratar muy brevemente.

Es indudable la necesidad de establecer un método semántico y gráfico que recoja de algún modo, en el esquema conceptual, el transcurso del tiempo y su influencia en la forma en que varían los datos. La aproximación más simple la constituyen atributos de tipo fecha que aparecen asociados a algunas entidades (véase la figura 2.50).

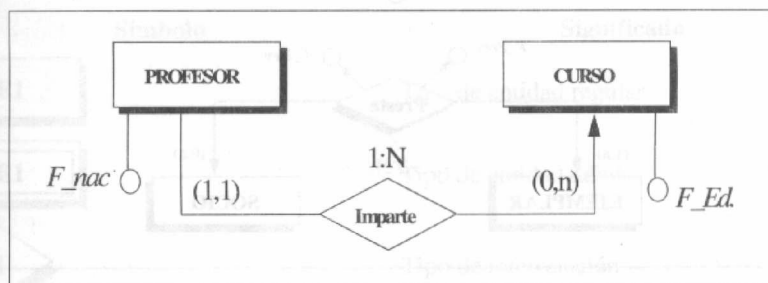


Figura 2.50. Ejemplo de entidades con atributos temporales

En este caso, la fecha de nacimiento de un profesor o la fecha en la que se impartió un curso son datos temporales recogidos en el esquema, pero se trata sólo de atributos que han de recibir un tratamiento especial en cuanto a las operaciones, y no se puede considerar realmente una aproximación semántica a la dimensión temporal.

Por otro lado, podemos analizar si los datos que se pretenden almacenar van a constituir una base de datos histórica o, si por el contrario, sólo nos interesa el estado actual de los mismos. La diferencia entre estos tipos de esquemas se puede apreciar en la figura 2.51 donde la parte superior se refiere a los préstamos actuales de libros en una biblioteca, de forma que una vez finalizado el préstamo la correspondiente información desaparece de la base de datos, sin que exista archivo histórico. En la parte inferior se representa el esquema conceptual de todos los préstamos que se han realizado en la biblioteca, recogiendo, además, el periodo de tiempo que duró el préstamo.

En caso de tratarse de datos históricos, los tipos de entidad o de interrelación correspondientes tendrán asociados siempre atributos de tipo *fecha*. Para sucesos puntuales, es decir, sin duración, bastará con un solo atributo de este tipo, mientras que para poder almacenar hechos que transcurren en un periodo de tiempo determinado necesitaremos una *fecha_inicio* y una *fecha_fin*.

En las bases de datos históricas en las que una interrelación entre dos ejemplares concretos se pueda repetir en el tiempo, el atributo fecha será multivaluado, como ocurre en la parte inferior de la figura 2.51, donde el mismo ejemplar se puede prestar al mismo socio en repetidas ocasiones.

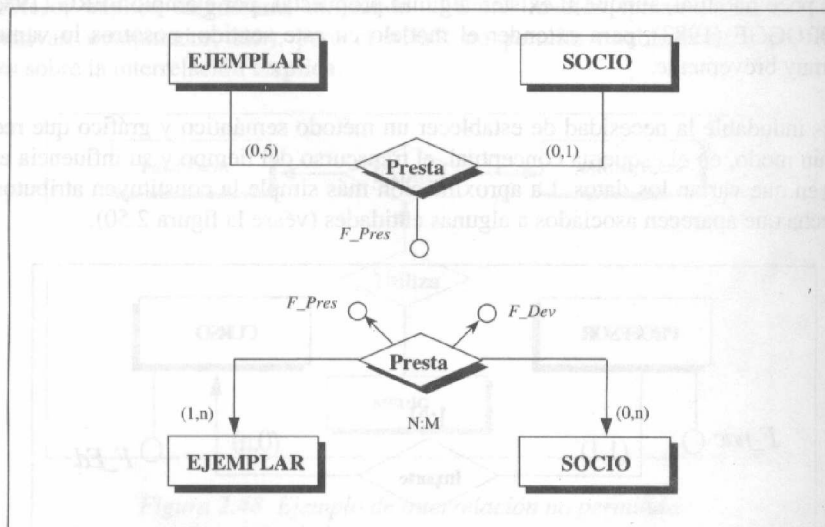


Figura 2.51. Introducción de la dimensión temporal en un esquema conceptual E/R

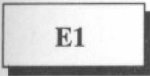
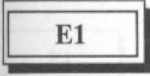
A veces resulta interesante representar la evolución de un tipo de entidad a lo largo del tiempo y aparece la noción de *estado*. Por ejemplo, si deseamos reflejar si un libro está en la biblioteca o se encuentra prestado, añadiremos al tipo de entidad un atributo que denominamos *estado*, que indicará en qué estado concreto se encuentra la entidad y que en muchos casos lleva asociado otro atributo, que es la fecha en la que se ha producido el cambio de estado; es también habitual en este tipo de aplicaciones que se desee tener constancia de la evolución de los estados, en cuyo caso se podría crear una nueva entidad, como *SITUACIÓN*, que tendría como atributos, entre otros posibles, *estado* y *fecha*. Observando el mundo real de los sistemas de información nos damos cuenta de que este mecanismo se utiliza sobre todo en la gestión de expedientes.

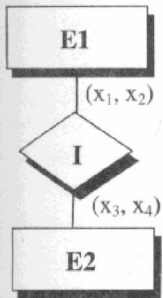
11. REPRESENTACIÓN GRÁFICA

En el capítulo anterior, al exponer qué es un modelo de datos, decíamos que una de las características de los modelos de datos es su forma de representación que puede ser en grafos o en tablas.

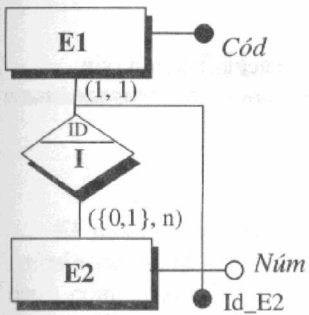
CHEN (1976), al presentar el ME/R, propone tanto una representación en grafos como en tablas, lo que se comprende si se tiene en cuenta que la finalidad del modelo era, como se indicaba en el mismo título del artículo, conseguir “una vista unificada de los datos”, por lo que no se podía limitar a un único tipo de diagramas. Sin embargo, la representación en tablas apenas ha tenido difusión en tanto que los grafos propuestos han tenido una amplia aceptación. Por esta razón nosotros sólo hemos considerado de interés la representación en grafos y la hemos ido realizando al tiempo que íbamos presentando cada uno de los elementos del modelo. Creemos, sin embargo, conveniente mostrar en su conjunto todas las convenciones que hemos utilizado, tal como se puede ver en el anexo a este capítulo.

ANEXO: SIMBOLOGÍA DEL MODELO ENTIDAD/INTERRELACIÓN

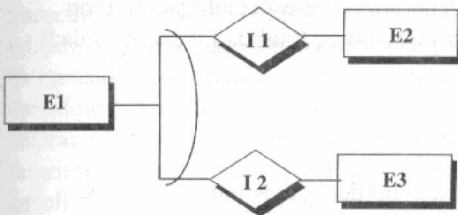
Símbolo	Significado
	Tipo de entidad regular
	Tipo de entidad débil
	Tipo de interrelación



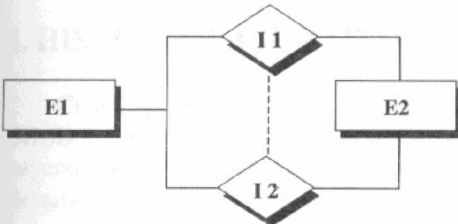
Cardinalidades de un tipo de interrelación



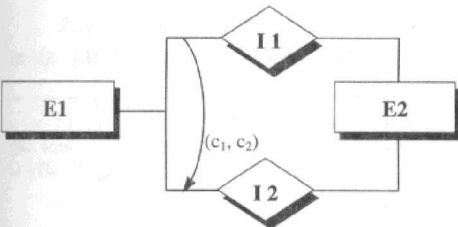
Tipo de interrelación (dependencia en identificación)



Exclusividad



Exclusión



Inclusividad



Tipo de interrelación (dependencia en existencia)

—○ Nombre

Atributo

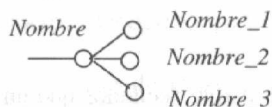
—● Nombre

Identificador Principal (IP)

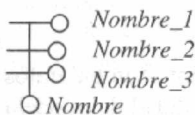
—◐ Nombre

Identificador Alternativo (IA)

Atributo compuesto



Representación 1



Representación 2

D
—○ Nombre

Atributo derivado
(D es la etiqueta que contendrá la fórmula de derivación)

(0,n)
---○ Nombre

Atributo opcional y multivaluado²¹ (sin restricciones)

(1,1)
—○ Nombre

Atributo obligatorio y univaluado²²

(0,1)
---○ Nombre

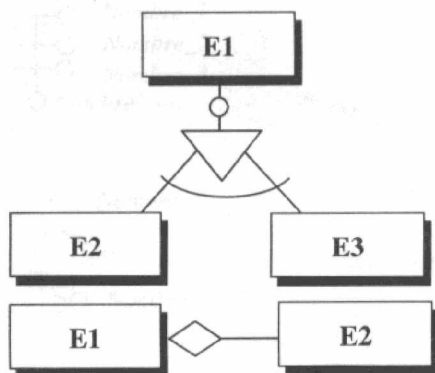
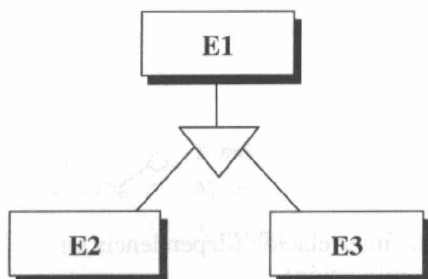
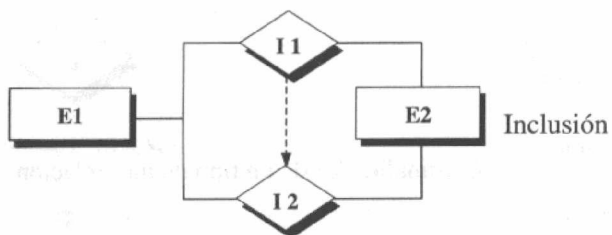
Atributo opcional y univaluado²¹

(1,n)
—>○ Nombre

Atributo obligatorio y multivaluado²¹

²¹ Se puede utilizar indistintamente la etiqueta o la simbología (línea de puntos y/o punta de flecha).

²² Puede no aparecer la etiqueta.



Ejemplo de agregación

CAPÍTULO 3

MODELO DE DATOS RELACIONAL

El modelo de datos relacional, presentado por Codd en 1970, en su célebre artículo de ACM titulado “Un modelo de datos relacional para grandes bancos de datos compartidos”, constituyó un hito en la historia de las bases de datos; historia cuya andadura se había iniciado hacía algo más de una década. En estos momentos, transcurridas casi tres décadas desde la publicación del artículo de Codd, los sistemas relacionales dominan el mercado y, según un estudio de IDC, en el año 1999 su cuota de mercado se acercará al 90% de las ventas mundiales de SGBD. Por esta razón, las metodologías de desarrollo de bases de datos, en su fase de diseño lógico, se suelen centrar en el modelo relacional. En este capítulo exponemos la estática del modelo relacional, en el marco formal de los modelos de datos del capítulo 1, insistiendo en aquellos conceptos, como es el de clave ajena, que más afectan al diseño¹.

1. HISTORIA Y OBJETIVOS

Cuando en el año 1970 el Dr. E. F. Codd propone un nuevo modelo de datos, los SGBD imperantes en el mercado, de tipo Codasyl y Jerárquico, no habían logrado superar el grave inconveniente que suponía la dependencia de las aplicaciones desarrolladas en ellos respecto a las estructuras de los datos.

A diferencia de estos modelos de datos basados en punteros físicos por los que tenía que navegar el programador a fin de recuperar y actualizar los datos, el Modelo

¹ Para una mayor profundización en el modelo de datos relacional o para el estudio de su parte dinámica (en la que aquí no entramos por estar este libro dedicado al diseño) se puede consultar DE MIGUEL y PIATTINI (1999), DATE (1995) y ELMASRI (1997).

Relacional (MR) se propone, como principal objetivo, aislar al usuario de las estructuras físicas de los datos, consiguiendo así la independencia de las aplicaciones respecto de los datos, finalidad perseguida desde los inicios de las bases de datos.

En el nuevo modelo, basado en la teoría matemática de las relaciones, los datos se estructuran lógicamente en forma de relaciones —tablas—. Esta formalización matemática convirtió rápidamente al modelo en una fuente fundamental de la investigación en bases de datos.

Los avances más importantes que el modelo de datos relacional incorpora respecto a los modelos de datos anteriores son:

- Sencillez y uniformidad: Los usuarios ven la base de datos relacional como una colección de tablas, y al ser la tabla la estructura fundamental del modelo, éste goza de una gran uniformidad, lo que unido a unos lenguajes no navegacionales y muy orientados al usuario final, da como resultado la sencillez de los sistemas relacionales.
- Sólida fundamentación teórica: Al estar el modelo definido con rigor matemático, el diseño y la evaluación del mismo puede realizarse por métodos sistemáticos basados en abstracciones.
- Independencia de la interfaz de usuario: los lenguajes relacionales, al manipular conjuntos de registros, proporcionan una gran independencia respecto a la forma en la que los datos están almacenados.

A pesar de que, desde su introducción, el modelo relacional se convirtió en uno de los principales temas de investigación en bases de datos, los primeros sistemas relacionales tardaron en aparecer, sobre todo por dificultades de implementación, y en general los sistemas de gestión de bases de datos existentes en el mercado no suelen recoger, incluso en la actualidad, algunas de las características y propiedades del modelo relacional teórico propuesto por Codd.

En el marco establecido en el capítulo 1, podemos definir el modelo relacional diferenciando su parte estática y su dinámica y, dentro de la primera, entre elementos permitidos y no permitidos (restricciones).

Puesto que el objetivo de este capítulo es establecer los fundamentos del modelo relacional a fines de diseño, sólo nos interesa la parte estática del modelo y, dentro de ella, aquellos aspectos más relacionados con el diseño relacional de bases de datos, como pueden ser, por ejemplo, las restricciones.

2. ELEMENTOS PERMITIDOS

La estructura básica, y única, del modelo relacional es la relación (también llamada tabla²), que sirve para representar tanto los objetos como las asociaciones entre ellos. Los *atributos* son las propiedades de las relaciones, y se definen sobre los *dominios*, los cuales, a diferencia de los atributos, tienen vida propia, es decir, existen con independencia de cualquier otro elemento del modelo, mientras que la existencia de un atributo va unida a la de la relación a la que pertenece.

2.1. Dominios, Relaciones y Atributos

El Universo del Discurso (UD) de una base de datos relacional está compuesto por un conjunto de dominios $\{D_i\}$ y de relaciones $\{R_i\}$ definidas sobre los dominios.

Un **dominio** es un conjunto nominado, finito y homogéneo³ de valores atómicos⁴. Cada dominio se especifica lógicamente mediante un nombre y un formato, el cual puede definirse por extensión (dando sus posibles valores) o por intensión (mediante un tipo de datos). A veces se asocia al dominio su unidad de medida (kilos, metros, etc.) y ciertas restricciones (como un rango de valores).

Por ejemplo, podemos definir el dominio **Materias**, cuyo conjunto de valores, definido por extensión, podría ser: Bases de Datos, Sistemas Operativos, Lenguajes, etc. Otro dominio podría ser **Códigos**, definido por intensión como carácter.

Un **atributo** (A) es la interpretación de un determinado dominio en una relación, es decir el “papel” que desempeña en la misma; si D es el dominio de A se denota:

$$D = \text{Dom}(A).$$

Por ejemplo, en la relación CURSO, un atributo puede ser *Cód_curso* y otro *Materia* definidos, respectivamente, sobre los dominios: **Códigos** y **Materias**.

Un atributo y un dominio pueden llamarse igual, pero hay que tener en cuenta que:

- Un atributo está siempre asociado a una relación, mientras que un dominio tiene existencia propia con independencia de las relaciones.

² Los SGBDR utilizan el término *tabla* en lugar de *relación* a fin de simplificar la nomenclatura de cara al usuario final; en realidad ambos términos no son sinónimos, como comentaremos posteriormente, aunque de hecho así se utilicen.

³ Decimos valores homogéneos porque son todos del mismo tipo.

⁴ Cada elemento de un dominio es indivisible en lo que respecta al modelo relacional; no puede, por tanto, ser a su vez una relación, ni un grupo repetitivo, etc.

- Un atributo representa una propiedad de una relación.
- Un atributo toma valores de un dominio.
- Varios atributos distintos (de la misma o de diferentes relaciones) pueden tomar sus valores del mismo dominio.

Matemáticamente, una **relación** definida sobre un conjunto de dominios $D_1 \dots D_n$ (no necesariamente distintos) es un subconjunto del producto cartesiano de los n dominios (n es el **grado** de la relación).

Podemos precisar mejor el concepto de relación si lo definimos en base a sus atributos, distinguiendo entre esquema de relación y relación⁵: un **esquema de relación** se compone de un nombre de relación R , de un conjunto de n atributos $\{A_i\}$ y de un conjunto de n dominios (no necesariamente distintos) $\{D_i\}$, donde cada atributo será definido sobre un dominio:

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

Una **relación** $r(R)$ es un conjunto de m elementos denominados tuplas $\{t_j\}$. Cada tupla _{j} es conjunto de pares $\langle A_1:v_{1j} \rangle, \dots \langle A_i:v_{ij} \rangle, \dots \langle A_n:v_{nj} \rangle$ donde cada A_i es el nombre de un atributo y v_{ij} es un valor del correspondiente dominio D_i sobre el que está definido el atributo:

$$r(R) = t_j \{ \langle A_1:v_{1j} \rangle, \dots \langle A_i:v_{ij} \rangle, \dots \langle A_n:v_{nj} \rangle : v_{ij} \in D_i \}$$

Una relación se representa utilizando una tabla donde:

- Las columnas de la tabla son los atributos que expresan las propiedades de la relación. El número de atributos se llama **grado** de la relación.
- Cada fila de la tabla, llamada tupla, es un elemento del conjunto que es la relación. El número de tuplas se llama **cardinalidad** de la relación. La cardinalidad varía en el transcurso del tiempo.

No se deben confundir los conceptos de tabla y de relación, puesto que:

- Una tabla es una forma de representar una relación.
- Una relación tiene unas propiedades intrínsecas que no tiene una tabla, y que se derivan de la misma definición matemática de relación, ya que, al tratarse de un conjunto, en una relación:

⁵ DATE (1995) distingue, por analogía con los lenguajes de programación, entre variable de relación y valor de una variable de relación.

- No puede haber dos tuplas iguales.
- El orden de las tuplas no es significativo.
- El orden de los atributos no es significativo.
- Cada atributo sólo puede tomar un único valor del dominio simple subyacente; no se admiten grupos repetitivos (ni otro tipo de estructuras) como valores de los atributos de una tupla.

3. INTENSIÓN Y EXTENSIÓN DE UNA RELACIÓN

De acuerdo con la definición de relación y esquema de relación dadas anteriormente, se pueden distinguir dos conceptos ligados a la noción de relación:

- *Intensión de una relación*: Parte definitoria y estática (invariante en el tiempo) de la relación, es lo que llamaremos *esquema de relación*.
- *Extensión*: Conjunto de tuplas que, en un instante determinado, satisfacen el esquema de relación y se encuentran almacenadas en la base de datos; es lo que se suele llamar, simplemente, relación. La extensión varía en el transcurso del tiempo.

A veces se utiliza el término relación en un sentido genérico para referirnos a un esquema de relación con sus posibles extensiones.

Estos dos conceptos están relacionados con la abstracción de clasificación. La intención se corresponde con el *tipo*, el cual se obtiene por clasificación de un conjunto de objetos (en este caso tuplas) en un ente de un nivel superior (más abstracto); cada tupla es un ejemplar del tipo, y la extensión en un determinado momento será el conjunto de ejemplares que, en ese momento, pertenecen al tipo y están almacenados en la base de datos.

En la figura 3.1 se puede ver un ejemplo de intención y extensión de una relación.

La intención de una base de datos relacional, llamada **esquema relacional**, está compuesta por una colección de esquemas de relación que describen un determinado universo del discurso; la extensión del esquema relacional, constituido por una colección de relaciones, es la base de datos relacional.

INTENSIÓN DE UNA RELACIÓN:CURSO (Cód_curso: **Códigos**, Nombre: **Nombres**, N_Horas: **Horas**, Materia: **Materias**)**EXTENSIÓN DE UNA RELACIÓN:**

CURSO

Cód_curso	Nombre	N_horas	Materia
00012	DISEÑO DE BASES DE DATOS	50	Bases de Datos
00034	BASES DE DATOS ORIENTADAS OBJETOS	30	Bases de Datos
00167	SISTEMAS OPERATIVOS AVANZADOS	30	Sis. Operativos
01521	ALMACENES DE DATOS	25	Bases de Datos
005142	INTRODUCCIÓN AL C++	25	Lenguajes

*Figura 3.1. Ejemplo de intención e extensión de una relación***4. ELEMENTOS NO PERMITIDOS: RESTRICCIONES**

Como en todo modelo de datos, los elementos no permitidos en la estructura del modelo relacional son de dos tipos: *restricciones inherentes* y *restricciones semánticas*.

4.1. Restricciones inherentes

Son las restricciones derivadas de la misma estructura del modelo, que no tienen que ser definidas por el usuario e imponen limitaciones a la hora de modelar nuestro mundo real.

La estructura del modelo relacional, al disponer solamente de un constructor, la relación, no permite diferenciar entre objetos y asociaciones entre ellos; distinción que suele ser habitual realizar en el mundo real y que resulta bastante evidente para los usuarios. Aun cuando algunos autores —por ejemplo, DATE (1995)— insisten en que tal distinción es arbitraria y subjetiva y que no hay razón alguna para tener dos conceptos distintos, la mayor parte de la literatura y de los modelos de datos semánticos prefieren mantener dos tipos de elementos para representar los objetos y las asociaciones. En definitiva, el modelo relacional impone limitaciones al modelado de datos debidas a la existencia de la relación como único constructor.

Otras restricciones inherentes del modelo relacional provienen de su misma definición matemática, ya que, al definirse la relación como un conjunto, no se permite la existencia de tuplas duplicadas (un conjunto no puede tener elementos iguales) y de ahí la *obligatoriedad* de una clave primaria o identificador (conjunto mínimo de atributos que identifican de forma unívoca las tuplas de una relación). No hay que confundir la obligatoriedad de que exista una clave primaria, que se deduce de una restricción inherente como es la prohibición de dos tuplas iguales, con la especificación de un conjunto de atributos como clave primaria de una relación, que es una restricción semántica (que veremos posteriormente) puesto que permite decir que un conjunto de atributos no puede tomar valores repetidos en una relación. También es preciso advertir que esta restricción inherente al modelo no suele ser de los productos, ya que, aunque en general permiten la definición de clave primaria, ésta no es obligatoria, es decir, se permiten tuplas duplicadas; lo mismo ocurre con el estándar SQL 92.

También la definición matemática de relación impone restricciones inherentes como la prohibición de que en el cruce de una fila y una columna haya más de un valor, es decir, las relaciones son tablas planas —de dos dimensiones— no admitiéndose los grupos repetitivos; ya hemos dicho que cada atributo sólo puede tomar un valor del dominio simple subyacente (es lo que se conoce como primera forma normal, como veremos al estudiar la teoría de la normalización). Además, el orden de las tuplas no es significativo (el orden de los elementos de un conjunto es irrelevante), por lo que operadores basados en un orden (que existen en otros modelos de datos⁶) no tienen razón de ser en el modelo relacional⁷.

Integridad de Entidad

Es una restricción inherente, debida asimismo a la necesidad de que todas las tuplas de una relación sean distintas, la cual establece que: “Ningún atributo que forme parte de la clave primaria puede tomar valores nulos”. Valor nulo de un atributo representa información desconocida, inaplicable, etc. Por lo tanto, si alguno de los atributos que forman parte de la clave primaria tomase valores nulos, algunas de las tuplas de la relación no podrían ser identificadas, por lo que se violaría la condición de que el valor de la clave debe ser único para cada tupla.

4.2. Restricciones semánticas

Las restricciones semánticas son facilidades proporcionadas por el modelo a fin de poder recoger en el esquema de relación la semántica del universo del discurso que

⁶ Por ejemplo, “FIND NEXT” (encontrar el siguiente) del modelo Codasyl.

⁷ Esto no quiere decir que una vez recuperado un conjunto de tuplas, éstas no se puedan presentar al usuario en un cierto orden lógico.

que afectan; posteriormente, lo haremos de acuerdo a la taxonomía expuesta en el capítulo 1, viendo tres tipos de restricciones:

- a) Condición y acción específicas
- b) Condición general y acción específica
- c) Condición general y acción general

4.2.1. CLASIFICACIÓN DE LAS RESTRICCIONES SEGÚN LOS ELEMENTOS A LOS QUE AFECTA LA CONDICIÓN

La condición de las restricciones semánticas en un modelo de datos hay que definirla siempre en función de los elementos del modelo, y así en el modelo relacional tendremos restricciones intrarrelaciones que afectan a los atributos de una sola relación (como la de clave primaria, unicidad, prohibición de nulos, etc.), interrelaciones que están definidas sobre los atributos de más de una relación (como la de integridad referencial), las que se definen sobre las tuplas de una relación, (por ejemplo, la cardinalidad) y las definidas sobre dominios (como una verificación sobre los valores de un dominio).

Por tanto, según los elementos sobre los que se defina la condición de las restricciones, éstas se pueden clasificar en:

- A) **Restricción intrarrelación sobre atributos:** Se define a nivel de esquema de relación, como el par $S1 = \langle A, P \rangle$ donde:

A es el conjunto de atributos del esquema de relación involucrados en la condición, denominado contexto de la restricción.

P es un predicado definido sobre **A**.

Entre estas restricciones están las de *unicidad* de uno o varios atributos cuyos valores no pueden repetirse en las distintas tuplas de una relación. En SQL 92:

UNIQUE <A>

donde **A** es el nombre de los atributos cuyos valores no han de repetirse en la relación **R**, en cuyo esquema se define la restricción; la cual se comprobará para operaciones (inserción de tuplas o modificaciones de algún valor de **A**) que puedan provocar el incumplimiento de la condición de unicidad, en cuyo caso se rechazará la operación a fin de evitar la inconsistencia.

Otra restricción de este tipo es la de *obligatoriedad* o prohibición de nulos en un atributo. En SQL92:

<nombre del atributo> NOT NULL

estamos modelando. Son restricciones que tiene que definir el diseñador a fin de que el esquema sea un reflejo lo más fiel posible del mundo real; de ahí la importancia de las restricciones en el diseño de BD. Es preciso advertir que las restricciones semánticas que ofrece el modelo relacional no siempre coinciden con las del SQL ni con las de los productos. Nosotros, para describirlas, nos apoyaremos en el SQL⁸, añadiendo los disparadores⁹.

En el modelo relacional, de forma análoga a lo que ocurre en general en un modelo de datos, una restricción de integridad es una regla ECA (Evento-Condición-Acción), donde el evento es una operación de actualización (inserción, borrado o modificación), la condición puede definirse como un predicado sobre un conjunto de atributos, de tuplas o de dominios, que debe ser verificado por los correspondientes elementos para que constituyan una extensión válida del esquema, y la acción puede ser de rechazo de la operación o cualquier otra, determinada por el modelo o bien por el usuario; la acción se lleva a cabo si una operación de actualización intenta violar la condición.

La implementación de las restricciones de integridad en un sistema relacional exige facilidades de definición de restricciones, así como procedimientos que impidan que en la base de datos aparezcan estados inconsistentes, es decir, procedimientos que pongan en marcha las correspondientes acciones a fin de mantener la consistencia.

Por tanto, un Sistema de Gestión de Bases de Datos Relacional (SGBDR), al igual que otros sistemas no relacionales, debe incluir facilidades que permitan:

En la fase de definición:

- Describir las restricciones con precisión y sencillez.
- Indicar las acciones ante una posible violación de una restricción.
- Verificar la consistencia de las restricciones entre sí mismas.

En fase de manipulación:

- Comprobar que las actualizaciones cumplen las restricciones de integridad.
- Poner en marcha las acciones indicadas en el caso de que las restricciones no se cumplan.

Vamos a describir las restricciones semánticas del modelo relacional dividiéndolas en primer lugar según una clasificación basada en los elementos a los

⁸ Para profundizar más en el SQL se puede consultar DE MIGUEL y PIATTINI (1999) y MELTON (1992).

⁹ Aunque en el SQL92 no existen los disparadores, éstos sí forman parte ya de varios productos.

La acción también es, en este caso, de rechazo.

Asimismo son restricciones de este tipo la definición de clave primaria, las restricciones de verificación (CHECK) y las aserciones sobre una única tabla (todas las cuales se verán posteriormente)

- B) **Restricción intrarrelación sobre tuplas:** Se define a nivel de esquema de relación, como el par $S2 = \langle T, P \rangle$ donde:

T es el conjunto de tuplas involucrados en la relación.

P es un predicado definido sobre **T**.

Por ejemplo, la comprobación de que la cardinalidad de una relación no sobrepasa un determinado valor, exigiría una condición de este tipo.

- C) **Restricción interrelación:** Se define en el esquema relacional, como el par $S3 = \langle A, P \rangle$ donde:

A es el conjunto de atributos pertenecientes a más de una relación, denominado contexto de la restricción.

P es un predicado definido sobre **A**.

Restricciones de este tipo son las claves ajenas y las aserciones que se definen sobre más de una tabla; en estas últimas, su intento de violación por una operación de actualización lleva consigo el rechazo de la operación, la cual no llega a llevarse a cabo.

- D) **Restricción sobre dominios:** Se define en el esquema relacional como el par $S4 = \langle D, P \rangle$ donde:

D es el conjunto de dominios involucrados en la restricción.

P es un predicado definido sobre **D**.

Una restricción de verificación sobre los valores de un dominio sería un ejemplo de este tipo de restricciones.

Además de las restricciones de rechazo (de las que acabamos de poner varios ejemplos), también los tipos de condiciones anteriores podrían servir para definir disparadores si, en lugar del rechazo, se especificase un procedimiento para la acción a disparar, aunque en este caso hay que tener en cuenta que la acción se dispararía cuando la condición se cumpliera.

En los ejemplos que acabamos de ver la condición está determinada en la propia restricción, por ejemplo, en el caso de obligatoriedad (NOT NULL) y de unicidad de los valores de un conjunto de atributos (UNIQUE); lo mismo ocurre en la definición de

clave primaria (PRIMARY KEY). Todas ellas son restricciones de condición específica y de acción específica, por lo que el diseñador se ha de limitar a poner el nombre de la restricción y los atributos a los que afecta. En el caso de las aserciones y de las restricciones de verificación, la condición es general y el diseñador tendrá que especificar el predicado concreto para cada una de ellas.

4.2.2. RESTRICCIONES DE CONDICIÓN Y ACCIÓN ESPECÍFICAS

Entre estas restricciones se encuentran, en el modelo relacional, además de las de unicidad y obligatoriedad ya vistas, las de claves candidata y ajena que veremos a continuación.

Clave candidata

Una clave candidata de una relación es un conjunto no vacío de atributos (descriptor) que identifican unívoca y mínimamente cada tupla de una relación.

En toda relación siempre hay, al menos, una clave candidata, ya que el conjunto de atributos que constituye la relación gozará de la propiedad de unicidad (por la definición misma de relación no puede haber dos tuplas iguales) y, si no tuviese la de minimalidad, se podría prescindir de aquellos atributos que lo impidieran, obteniendo así una clave candidata.

Una relación puede tener más de una clave candidata, entre ellas cabe distinguir:

- *Clave primaria:* Es la clave candidata que el usuario elegirá, por consideraciones ajenas al modelo relacional, para identificar las tuplas de la relación. Los atributos que forman parte de la clave primaria no pueden tomar valores nulos (recuérdese la restricción inherente que es la integridad de entidad).
- *Claves alternativas:* Son aquellas claves candidatas que no han sido elegidas como claves primarias de la relación.

El SQL 92 (así como la mayor parte de los productos) proporciona un mandato para la creación de la clave primaria de una relación (sólo puede haber una en cada relación):

PRIMARY KEY <nombre de los atributos que forman la clave primaria>.

Para la definición de un conjunto de atributos como clave alternativa existe el mandato UNIQUE (al que ya nos hemos referido); y si, además, se desea que los nulos no estén permitidos, se puede imponer la restricción de obligatoriedad (NOT NULL).

En el caso de la clave primaria, la definición de la misma es equivalente a las restricciones de obligatoriedad y unicidad

Por ejemplo, en la tabla de la figura 3.1 elegimos como clave primaria el *Código* de curso y, suponiendo que los nombres no se repitiesen, el *Nombre* sería una clave alternativa.

Clave ajena¹⁰

Una clave ajena de una relación R2 es un conjunto no vacío de atributos cuyos valores han de coincidir con los valores de la clave primaria¹¹ de una relación R1 (R1 y R2 pueden ser la misma relación). Se dice que R2 es la relación *que referencia*, mientras que R1 es la relación *referenciada*.

En la figura 3.2 se muestra un ejemplo de clave ajena, donde los valores del atributo *Cód_prog* de la relación CURSO_DOCTORADO (relación que referencia) deben coincidir con los que es la clave primaria de la relación PROGRAMA (relación referenciada).

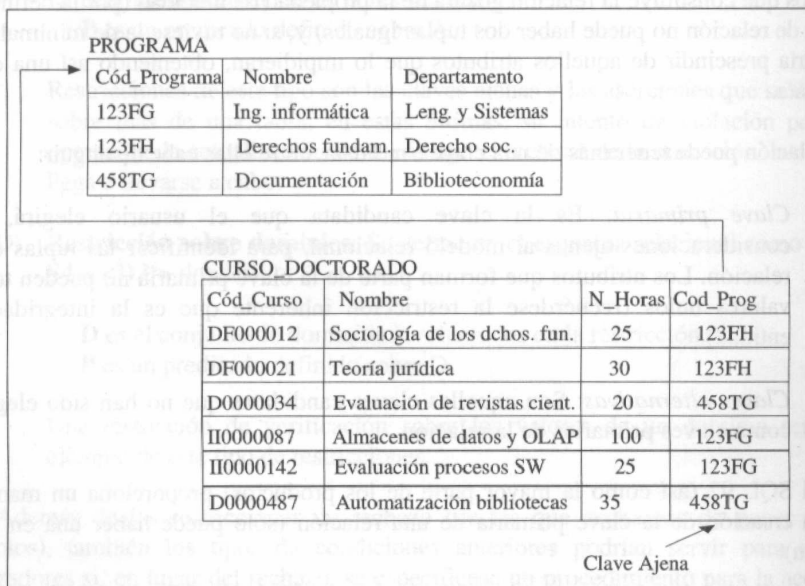


Figura 3.2. Ejemplo de clave ajena

¹⁰ También llamadas "externas" o "foráneas" (Foreign key, en inglés)

¹¹ El SQL92 no exige que sea una clave primaria, sino que admite cualquier clave candidata.

Integridad referencial

Dentro de las restricciones propias del modelo relacional se encuentra la de integridad referencial, que es de condición y acción específicas y que afirma: “Si una relación R2 tiene un descriptor CA que es clave ajena que referencia a la clave primaria¹² CP de la relación R1, todo valor de CA debe coincidir con un valor de CP, o ser nulo”. Ésta es la condición de la restricción, la cual puede expresarse como un predicado:

$$R2.CA = R1.CP$$

Los descriptores CA y CP han de estar definidos sobre el mismo dominio y se permite que sobre CA se defina, si es necesario, la restricción de obligatoriedad (si no se define, la clave ajena admitirá valores nulos¹³).

En cuanto a la acción, es de tipo específico (aunque no es de rechazo en todos los casos). Si se intenta insertar una tupla en la tabla que referencia R2, que no cumpla la condición, la acción es de rechazo. Si la condición falla debido a una operación de borrado de tuplas o de modificación de la clave primaria en la tabla referenciada R1, existe en SQL92 la posibilidad de elegir entre cuatro opciones, tanto para la operación de borrado como para la de modificación:

- NO ACTION (rechazar la operación)
- CASCADE (propagar la modificación o borrar las tuplas de la tabla que referencia)
- SET NULL (poner valor nulo en la CA de la tabla que referencia)
- SET DEFAULT (poner valor por defecto en la CA de la tabla que referencia)

La primera se toma por omisión y supone el rechazo de la correspondiente operación de borrado o de modificación, la cual no llega a llevarse a cabo. En la segunda se borran (o modifican) todas las tuplas de la tabla que referencia cuyo valor de la clave ajena coincide con el valor de la clave primaria que se borra (o se modifica). En la opción SET NULL se pone valor nulo en todos los atributos que forman parte del descriptor que constituye la clave ajena, cuyo valor ha desaparecido en la clave primaria de la tabla referenciada. En el último caso se pone el valor que se ha definido por defecto para la clave ajena en la tabla que referencia.

En la figura 3.3 aparecen algunos ejemplos de definición de claves ajenas con sus opciones.

¹² Téngase en cuenta lo dicho anteriormente de que el SQL92 admite que en lugar de una clave primaria sea cualquier clave candidata.

¹³ Obsérvese que es la clave ajena en la tabla que referencia la que admitiría valores nulos, nunca la clave primaria en la tabla referenciada.

B) Restricción de aserción

Supongamos, siguiendo el mismo ejemplo, que en la relación CONCEDE entre BECA y ESTUDIANTE se desea indicar que solamente se pueda conceder una beca a uno de los estudiantes que ha solicitado dicha beca¹⁴; para conseguirlo, se podría definir la siguiente aserción en el esquema relacional:

```
CREATE ASSERTION CONCEDE_SOLICITA AS
CHECK (SELECT Cod_Estudiante, Cod_Beca
FROM CONCEDE) IN
(SELECT Cod_Estudiante, Cod_Beca
FROM SOLICITA));
```

Esta aserción sustituiría la clave ajena que habríamos definido, ya que haría exactamente lo mismo que hace la clave ajena: impedir que cualquier tipo de actualización deje en la relación CONCEDE tuplas que no estaban en SOLICITA.

4.2.4. RESTRICCIONES DE CONDICIÓN GENERAL Y ACCIÓN GENERAL (DISPARADORES)

El SQL92 no soporta restricciones en las que el usuario pueda especificar libremente la respuesta (acción) ante una determinada condición, lo que se suele llamar **disparadores**¹⁵ (trigger). Así como las anteriores reglas de integridad son *declarativas*, los disparadores son *procedimentales* (al menos en lo que a la acción se refiere), siendo preciso que el usuario escriba el procedimiento que ha de aplicarse en caso de que se cumpla la condición¹⁶. Aunque el SQL92 no soporta disparadores, éstos sí se encuentran en muchos productos, así como en el estándar SQL:1999.

Este tipo de restricciones es muy importante en el diseño de bases de datos a fin de no perder en el esquema relacional aquella semántica del esquema conceptual que no es posible recoger mediante las anteriores restricciones.

Para ilustrar el uso de disparadores, siguiendo el mismo ejemplo, vamos a suponer que si una beca es solicitada por más de 50 alumnos, se introduce un texto en una tabla de mensajes para que, la persona que gestiona las becas, considere si es necesario ofrecer más becas.

¹⁴ Es decir, existe una **restricción de inclusión** entre CONCEDE y SOLICITA, ya que CONCEDE ha de ser un subconjunto de las tuplas de SOLICITA.

¹⁵ Los disparadores no sólo se utilizan para garantizar la integridad de la base de datos, sino que tienen también un importante campo de aplicación en las bases de datos activas.

¹⁶ Obsérvese que en los disparadores la acción se activa cuando la condición se cumple, al contrario de lo que ocurre en otro tipo de restricciones.

```
CREATE TRIGGER Comprobar_Matriculados
AFTER INSERT ON SOLICITA
DECLARE
    NUM_SOLICITUDES Number;
BEGIN
    SELECT COUNT(*) INTO NUM_SOLICITUDES FROM SOLICITA;
    IF NUM_SOLICITUDES > 50 THEN
        INSERT INTO MENSAJES VALUES ('Hay más de 50 solicitudes');
    END IF;
END Comprobar_Matriculados;
```

5. LOS TRES NIVELES DE ANSI EN EL MODELO RELACIONAL

El modelo relacional responde a la parte lógica de la arquitectura a tres niveles de ANSI. El esquema conceptual de ANSI será el *esquema relacional* y los esquemas externos se corresponden con las *vistas*. El modelo relacional es un modelo lógico y, por tanto, no contempla los aspectos físicos del nivel interno de ANSI, que son propios de los productos; por ello, el SQL92 no tiene ninguna sentencia de tipo físico como, por ejemplo, "CREATE INDEX".

5.1. El nivel conceptual del modelo relacional: Esquema de Relación y Esquema Relacional

El *esquema de relación* como intensión de una relación, es una descripción de la misma, donde, además de los atributos (con referencia a los correspondientes dominios) sobre los que está definida la relación, es preciso especificar también las restricciones de integridad que deben cumplir las tuplas de la relación para ser ejemplares válidos de dicho esquema.

Por tanto, un esquema de relación se define como:

R (A:D, S)

donde:

R es el nombre de la relación.

A es el conjunto de atributos, cada uno de los cuales está definido sobre un dominio **D**

S son las restricciones de integridad intrarrelación (sobre atributos o sobre tuplas)

Una extensión válida de un esquema de relación es una *relación base*, definida sobre el conjunto de atributos de dicho esquema, donde cada atributo toma sus valores del correspondiente dominio y que satisface todas sus restricciones de integridad.

El *esquema relacional*, como intensión de una base de datos relacional, es una descripción de la misma donde, además del conjunto de esquemas de relación, es preciso especificar los dominios y las restricciones de integridad interrelación y sobre dominios, además de las vistas.

Por tanto, un esquema relacional se define como:

$E(R, D, T, V)$

donde:

E es el nombre del esquema relacional

R es el conjunto de esquemas de relación.

D es la definición del conjunto de dominios.

T es el conjunto de restricciones de integridad interrelación y sobre dominios.

V es el conjunto de vistas.

Una extensión del esquema relacional es el conjunto de valores de los dominios que forman parte del esquema relacional, más el conjunto de extensiones de los esquemas de relación que lo componen y que satisfacen todas sus restricciones (conjunto de relaciones base).

5.2. Las vistas y el nivel externo en el modelo relacional

El nivel externo en el modelo relacional está constituido, además de por las relaciones base, por las *vistas*, las cuales son relaciones derivadas que se definen dando un nombre a una expresión de consulta. Se podría decir que las vistas son *relaciones virtuales* (como “ventanas” sobre otras relaciones), en el sentido de que no tienen datos almacenados, sino que lo único que se almacena es su definición en términos de otras relaciones.

Las vistas son importantes en el diseño de bases de datos porque proporcionan a los usuarios una forma de ver los datos más sencilla y apropiada a las necesidades de sus aplicaciones. Las vistas facilitan además la definición de restricciones de confidencialidad.

Una diferencia importante entre la arquitectura ANSI y el modelo relacional es que en éste la visión del usuario no está limitada a una vista, sino que un esquema externo de ANSI en el modelo relacional puede estar formado por un conjunto de vistas y/o de tablas base, es decir, los usuarios pueden “ver” directamente las tablas base, además (o en lugar) de las vistas.

Las vistas se definen mediante una sentencia del lenguaje de definición de datos (“CREATE VIEW”) que contiene una expresión de consulta, la cual no se ejecuta en el momento de su definición, sino posteriormente cuando se invoca la vista al aparecer su nombre en una consulta (en una sentencia “SELECT”) o en una actualización (inserción -“INSERT”-, borrado -“DELETE”- o modificación -“UPDATE”-)¹⁷.

5.3. El nivel interno en el modelo relacional

El nivel interno de una base de datos describe cómo se encuentran los datos almacenados en el soporte físico. Es un objeto prioritario del diseño físico minimizar el número de accesos al soporte donde se encuentran los datos almacenados físicamente.

El modelo relacional es un modelo lógico y, por tanto, como ya hemos señalado anteriormente, sólo contempla los dos niveles lógicos: el conceptual (relaciones base) y el externo (relaciones base y vistas), por lo que no se puede hablar propiamente del nivel interno en el modelo relacional. Sin embargo, sí es preciso que los SGBDR tengan los datos físicamente almacenados en memoria secundaria (no sólo las tablas sino también las estructuras complementarias, como índices, punteros, direcciones de páginas, etc., a fin de conseguir un acceso más eficiente); la descripción de todos estos datos almacenados constituye el esquema interno de la arquitectura ANSI.

5.4. Correspondencia de la arquitectura ANSI y el modelo relacional

En la figura 3.4 se puede observar, como resumen de lo que acabamos de exponer, la correspondencia entre los tres niveles de la arquitectura ANSI y el modelo relacional.

¹⁷ Es preciso tener en cuenta que las sentencias de actualización se pueden aplicar a las vistas (aunque sólo a las vistas que provienen de una sola tabla), pero ello no quiere decir que se actualice la vista, sino la tabla de la cual procede. Aunque, teóricamente, existen cierto tipo de vistas procedentes de más de una tabla a las que podrían aplicarse sentencias de actualización; sin embargo, en la práctica, los productos sólo dejan aplicar sentencias de actualización a las vistas que provienen de una única tabla.

ANSI		RELACIONAL	
L O G I C O	Nivel Externo	S Q L	Vistas
	Nivel Conceptual		Relaciones Base
F I S I C O	Nivel Interno	P R O D U C T O S	Datos Almacenados - Relaciones base almacenadas - Índices - Punteros - Direcciones de página - ...

Figura 3.4. ANSI y el Modelo Relacional

KVS@C

- 4. Concepto y manipulación de dependencias funcionales**
- 5. Teoría de la normalización:
Formas normales basadas en las dependencias funcionales**
- 6. Formas normales avanzadas y reorganización de relaciones**
- 7. Algoritmos de diseño en el modelo relacional**

***P
A
R
T
E

II***

CONCEPTO Y MANIPULACION DE DEPENDENCIAS FUNCIONALES

La idea central en el diseño formal de una base de datos directamente en el modelo relacional es el concepto de dependencias entre los datos. Intimamente ligado con este concepto de dependencias, tenemos el de Forma Normal en la que se encuentra un esquema de relación. En este capítulo, después de analizar el concepto de dependencias entre los datos y de ver los distintos tipos de dependencias, desarrollamos el concepto de dependencia funcional, la implicación lógica de dependencias funcionales y los axiomas de Armstrong que permiten la derivación de nuevas dependencias, para terminar definiendo diversos conceptos relacionados con la teoría de las dependencias como son el de cierre transitivo de un descriptor, el de cierre de un conjunto de dependencias, equivalencia de dos conjuntos de dependencias, etc.; conceptos imprescindibles para la aplicación de la teoría de la normalización.

1. DEPENDENCIAS ENTRE LOS DATOS

Las dependencias son propiedades inherentes al contenido semántico de los datos; forman parte de las restricciones de usuario del modelo relacional y se han de cumplir para cualquier extensión de un esquema de relación.

Puesto que las dependencias constituyen una parte importante de la semántica del Universo del Discurso que se pretende modelar, son incluidas, en la definición de esquema relacional y de los esquemas de relación, que forman parte del mismo. A fines de simplificación, y sin perder por ello generalidad, para el estudio de las dependencias vamos a considerar que el esquema relacional está compuesto por un único esquema de relación, el cual es un par de la forma:

R (A,DEP)

donde:

A es el conjunto de atributos de la relación¹.

DEP es el conjunto de dependencias existentes entre los atributos.

Existen distintos tipos de dependencias: **funcionales, multivaluadas, jerárquicas y de combinación (también llamadas producto)**. Cada tipo de dependencia se caracteriza por ser una asociación particular entre los datos. Además, cada tipo de dependencia constituye un caso particular del grupo que le sigue; así, las dependencias funcionales son un caso particular de las dependencias multivaluadas, y así sucesivamente (véase figura 4.1).

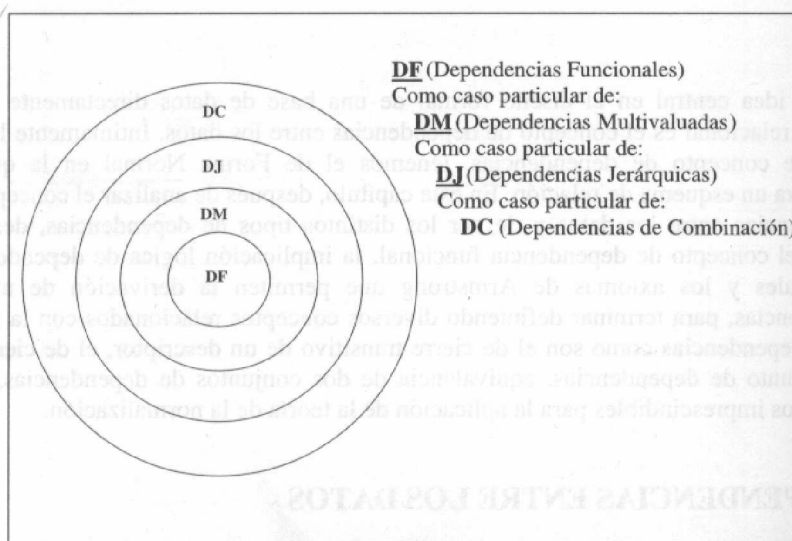


Figura 4.1. Distintos tipos de Dependencias

Aun cuando cada tipo de dependencia tenga una serie de características particulares, todas ellas presentan aspectos comunes: velos dependencias.

- Son invariantes en el tiempo, siempre que no varíe el mundo real que se está modelando; por lo tanto, se han de cumplir para cualquier extensión del esquema de relación.

¹ Prescindimos de los dominios para no complicar la notación, y porque con ello no perdemos generalidad.

- Son propiedades inherentes al contenido semántico de los datos y forman parte de las restricciones de usuario del modelo relacional.
- No es posible deducir la existencia de una dependencia a partir de la observación de una extensión del esquema de relación; la existencia de las dependencias viene determinada únicamente por la semántica del Universo del Discurso que se pretende modelar. Dicho de otra forma, las dependencias no recogen propiedades que se cumplen “por casualidad” en un cierto momento; por ejemplo, el que todos los alumnos de doctorado hayan recibido este año notas distintas, es decir, que este año una determinada nota sólo le pueda corresponder a un alumno, no es una propiedad general que se ha de cumplir para cualquier extensión de la base de datos, ya que en otra extensión (cualquier otro año) dos alumnos pueden tener la misma nota, por tanto ésta no sería una restricción (dependencia) del esquema, sino algo que se cumple “por casualidad” este año.
- A partir de una extensión válida de un esquema de relación sí será posible comprobar que una dependencia no se cumple para ese esquema, ya que las dependencias se han cumplido siempre, para cualquier extensión.

El grupo más restrictivo (y también más numeroso) de asociaciones entre los datos es el de las dependencias funcionales. Sobre este conjunto de dependencias se apoyan las tres primeras formas normales y la forma normal de Boyce_Codd.

2. CONCEPTO DE DEPENDENCIA FUNCIONAL

Definición I:

Sea el esquema de relación $R(A,DF)$, y sean X, Y dos subconjuntos de A , a los que llamamos **descriptores**. Se dice que **Y depende funcionalmente de X o que X implica o determina a Y** , y se denota como $X \rightarrow Y$ si, y sólo si, a cada valor x del atributo X , le corresponde un único valor y del atributo Y .

Se puede definir también, más formalmente, el concepto de dependencia funcional de la siguiente forma:

Definición II:

Sea el esquema de relación $R(A,DF)$ y sean X e Y dos **descriptores**. Se dice que **existe una dependencia funcional entre X e Y** de forma que X determina a Y , si, y sólo si, se cumple que para cualesquiera dos tuplas de R , u y v tales que $u[X] = v[X]$, entonces necesariamente $u[Y] = v[Y]$ (la notación $u[X]$ o $v[X]$ indica la proyección de la tupla u o de la tupla v , respectivamente, sobre el descriptor X , cuyo resultado será

un valor de X); análogamente $u[Y]$ o $v[Y]$ indica la proyección de u o de v , respectivamente, sobre el descriptor Y .

Un **determinante** o **implicante** es un conjunto de atributos del que depende funcionalmente otro conjunto de atributos al que llamamos **implicado**.

✓ Por ejemplo, podemos decir que el Código de estudiante determina el nombre del mismo:

Cód_Estudiente $\rightarrow$ *Nombre*

El *Cód_Estudiente* es el determinante o implicante y el *Nombre* es el implicado.

Dos descriptores X e Y se dice que son equivalentes si:

$$X \rightarrow Y \wedge Y \rightarrow X$$

lo que también se puede representar:

$$X \leftrightarrow Y$$

Por ejemplo, los atributos *Cód_Estudiente* y *DNI* son equivalentes (se supone que dos alumnos distintos no pueden tener ni el mismo código ni el mismo DNI), es decir:

$$Cód_Estudiante \leftrightarrow DNI$$

Queremos llamar la atención sobre la afirmación de que *Cód_Estudiente* determina *DNI* no significa que, conocido el código de un alumno, podamos deducir, a partir de él, cuál es su DNI, a no ser que tengamos la extensión r del esquema de relación que contiene la correspondiente dependencia funcional. Es decir, si para un esquema R , tenemos la dependencia funcional:

$$X \rightarrow Y$$

Dado un valor x de X no podemos, en general, conocer el valor y de Y , solamente nos limitaremos a afirmar que, para dos tuplas, de cualquier extensión $r(R)$ que tengan el mismo valor de X , el valor de Y será también igual en ambas. Sin embargo, puede ocurrir, a veces, que Y sea un descriptor *derivado* de X por cualquier expresión aritmética o lógica, en cuyo caso el conocimiento de un valor x de X nos lleva a conocer el correspondiente valor y de Y (pero téngase en cuenta que se trata de un caso particular que no se cumple para cualquier dependencia funcional); por tanto, todo atributo derivado depende funcionalmente del atributo (o atributos) del que se deriva, pero este tipo de dependencias no plantea problemas de inconsistencia (si realmente el atributo derivado se calcula automáticamente) y, por tanto, no es necesario tenerlo en cuenta al aplicar la teoría de la normalización.

Una herramienta muy útil a la hora de explicitar las dependencias funcionales es el grafo o diagrama de dependencias funcionales, mediante el cual se representa un conjunto de atributos y las dependencias funcionales existentes entre ellos.

En el grafo aparecen los nombres de los atributos unidos por flechas, las cuales indican las dependencias funcionales y parten del implicante hacia el implicado. Cuando el implicante de una dependencia no es un único atributo, es decir, se trata de un implicante compuesto, los atributos que lo componen se encierran en un recuadro y la flecha parte de éste, no de cada atributo.

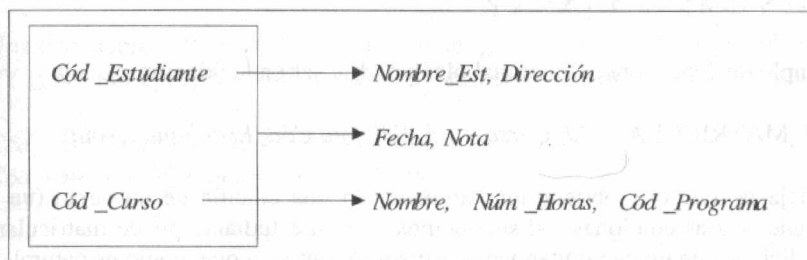


Figura 4.2. Ejemplo de diagrama de dependencias funcionales

En la figura 4.2 se presenta un ejemplo de cómo se visualizan las dependencias; podemos observar que *Cód_Estudiante* determina funcionalmente el *Nombre* y la *Dirección*, como indica la correspondiente flecha; de forma análoga, *Cód_Curso* determina el *Nombre* del curso, el *Núm_Horas* y el *Programa*; mientras que en conjunto *Cód_Estudiante* y *Cód_Curso* (lo que se indica mediante el recuadro que los incluye) determinan la *Fecha* y la *Nota*.

Otra forma de representar un conjunto de dependencias funcionales aparece en la figura 4.3.

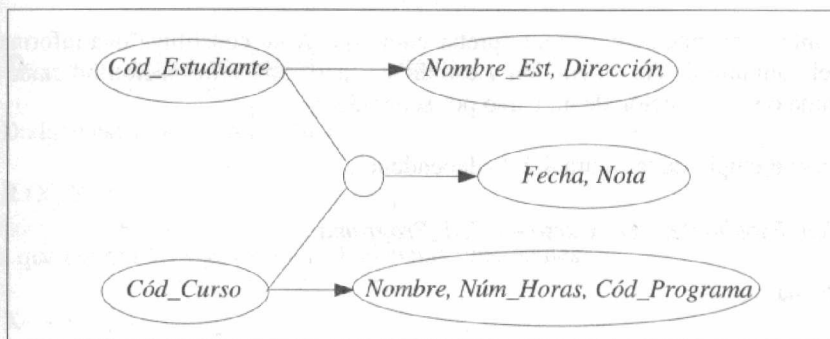


Figura 4.3. Ejemplo de otra forma de representar las dependencias funcionales

2.1. Dependencia funcional plena o completa

Se dice que Y tiene *dependencia funcional completa o plena* de X , si depende funcionalmente de X , pero no depende de ningún subconjunto de éste.

Se representa por $X \Rightarrow Y$.

Por tanto,

$$X \Rightarrow Y \text{ sii } \nexists X' \subset X \mid X' \rightarrow Y.$$

Un ejemplo de dependencia funcional plena podría ser en la relación:

SE_MATRICULA (*Cód_Curso*, *Cód_Edición*, *Cód_Estudiante*, *Nota*)

que refleja la nota que obtiene un estudiante en una edición de un curso (un curso puede tener varias ediciones); si suponemos que un estudiante puede matricularse en varias ediciones de un curso y en varios cursos distintos, y que, como es natural, en un curso se matriculan varios estudiantes, se da la siguiente dependencia funcional:

$$Cód_Curso, Cód_Edición, Cód_Estudiante \rightarrow Nota$$

Sin embargo:

$$\nexists X' (Cód_Curso, Cód_Edición, Cód_Estudiante) \mid X' \rightarrow Nota$$

por lo tanto, *Nota* depende funcionalmente de forma completa de *Cód_Curso*, *Cód_Edición*, *Cód_Estudiante*, esto es:

$$Cód_Curso, Cód_Edición, Cód_Estudiante \Rightarrow Nota$$

lo que intuitivamente se puede interpretar como que *Nota* constituye una información sobre el conjunto de curso, edición y estudiante; pero esta información no atañe a un estudiante o a una edición de un curso por separado.

En el ejemplo de la figura 4.2, la dependencia:

$$Cód_Estudiante, Cód_Curso \rightarrow Cód_Programa$$

no es plena, ya que:

$$Cód_Curso \rightarrow Cód_Programa$$

por tanto:

$$\text{Cód_Estudiante, Cód_Curso} \not\rightarrow \text{Cód_Programa}$$

se dice que, en esta dependencia, *Cód_Estudiante* es un atributo **redundante** o **ajeno** a la dependencia, también llamado **extraño**.

2.2. Dependencia funcional trivial

Una dependencia funcional $X \rightarrow Y$ se dice que es *trivial* si Y es un subconjunto de X ($Y \subseteq X$). Por ejemplo, serán triviales las siguientes dependencias:

$$\begin{aligned} \text{Cód_Estudiante} &\rightarrow \text{Cód_Estudiante} \\ \text{Cód_Curso, Cód_Edición} &\rightarrow \text{Cód_Curso} \end{aligned}$$

2.3. Dependencia funcional elemental

Decimos que una dependencia funcional $X \rightarrow Y$ es **elemental** si Y es un atributo **único** no incluido en X , y no existe X' incluido en X tal que $X' \rightarrow Y$, es decir, la dependencia funcional elemental es una dependencia funcional completa, no trivial y en la que el implicado es un atributo único:

$$X \rightarrow Y \text{ es elemental sii } (\nexists Y' \subset Y) \wedge (Y \subseteq X) \wedge (\nexists X' \subset X \mid X' \rightarrow Y)$$

No todas las dependencias funcionales son útiles para la teoría de la normalización, sino solamente las elementales, que acabamos de definir, y que son un subconjunto de las dependencias funcionales.

2.4. Dependencia funcional transitiva

Dado el esquema de relación

$$R(X, Y, Z)$$

en las que existen las siguientes dependencias funcionales:

$$\begin{aligned} X &\rightarrow Y \\ Y &\rightarrow Z \\ Y &\not\rightarrow X \end{aligned}$$

se dice que Z tiene una **dependencia transitiva** respecto a X a través de Y , lo que se representa por

$$X \twoheadrightarrow Z$$

Si, además, $Z \not\rightarrow Y$ se dice que la dependencia transitiva es **estricta**.

Si consideramos la relación

CURSO_PROGRAMA(*Cód_Curso*, *Cód_Programa*, *Cód_Departamento*)

en donde tenemos para cada curso su código, el programa que lo incluye y el departamento del que depende el programa (suponiendo que un curso se imparte en un único programa y que un programa lo prepara un único departamento) se tendrán las siguientes dependencias funcionales:

$$Cód_Curso \rightarrow Cód_Programa$$

$$Cód_Programa \rightarrow Cód_Departamento$$

Además:

$$Cód_Programa \not\rightarrow Cód_Curso \quad (\text{ya que en un programa se imparten varios cursos})$$

Entonces:

$$Cód_Curso \rightarrow Cód_Departamento$$

por tanto, la dependencia funcional entre *Cód_Curso* y *Cód_Departamento* es una dependencia transitiva a través de *Cód_Programa*, que se representa por:

$$Cód_Curso \twoheadrightarrow Cód_Departamento$$

(lo que se puede interpretar intuitivamente como que *Cód_Curso* es una información sobre el curso, pero indirectamente, ya que constituye una información sobre el programa y éste, a su vez, sobre el curso)

Además, como *Cód_Departamento* $\not\rightarrow$ *Cód_Programa*, se da una dependencia transitiva estricta.

En cambio, si tuviéramos la relación:

CURSO (*Cód_Curso*, *Nombre*, *Cód_Programa*)

con las siguientes dependencias:

- $Cód_Curso \rightarrow Nombre$
- $Nombre \rightarrow Cód_Curso$
- $Cód_Curso \rightarrow Cód_Programa$
- $Nombre \rightarrow Cód_Programa$

Ninguna de las dependencias $Cód_Curso \rightarrow Cód_Programa$ ni $Nombre \rightarrow Cód_Programa$ son transitivas, al ser los atributos $Cód_Curso$ y $Nombre$ equivalentes.

3. IMPLICACIÓN LÓGICA DE DEPENDENCIAS FUNCIONALES Y AXIOMAS DE ARMSTRONG

El conocimiento de ciertas dependencias funcionales nos puede llevar a inferir la existencia de otras que no se encontraban en el conjunto inicial, es decir, dado un esquema de relación:

$R(A, DF)$

es posible deducir de DF nuevas dependencias que sean una consecuencia lógica del conjunto de partida. Sea, por ejemplo, la relación

⑧ SOLICITA ($\{Cód_Estudiante, DNI_Est, Cód_Beca, Fecha_Solicitud\}, DF$)

donde:

$$DF = \{ \begin{array}{l} Cód_Estudiante \rightarrow DNI_Est \\ DNI_Est \rightarrow Cód_Estudiante \\ Cód_Estudiante, Cód_Beca \rightarrow Fecha_Solicitud \end{array} \}$$

Podemos deducir lógicamente de estas dependencias que, dado el DNI de un estudiante y un código de beca, la fecha de solicitud está determinada, es decir:

$$DNI_Est, Cód_Beca \rightarrow Fecha_Solicitud$$

La dependencia anterior se cumple para cualquier extensión r de R y decimos que es una **consecuencia lógica** de DF (o que viene implicada por DF), lo que expresamos:

$$DF \models DNI_Est, Cód_Beca \rightarrow Fecha_Solicitud$$

3.1. Consecuencia lógica y derivación de dependencias funcionales

Dado el esquema de relación $R(A, DF)$, una dependencia funcional f es una consecuencia lógica de DF (expresado $DF \models f$) si f se cumple para cualquier extensión r de R .

El **cierre** de un conjunto de dependencias funcionales DF (que se denota DF^+) es el conjunto de todas las dependencias que son consecuencia lógica de DF :

$$DF^+ = \{ X \rightarrow Y \mid DF \models X \rightarrow Y \}$$

DF será siempre un subconjunto del cierre ($DF \subseteq DF^+$). Por lo tanto, las notaciones $R(A, DF)$ y $R(A, DF^+)$ definen el mismo esquema de relación.

Sin embargo, las definiciones de consecuencia lógica y de cierre de un conjunto de dependencias funcionales no permiten el cálculo de éste, por lo que se hacen necesarias unas reglas de derivación que faciliten la implicación lógica de dependencias.

Estas reglas de inferencia, mediante las cuales se pueden derivar dependencias a partir de un conjunto inicial, constituyen, como se demuestra en ARMSTRONG (1974), un conjunto completo y correcto de axiomas que se conocen con el nombre de **Axiomas de Armstrong**.

Dado un conjunto DF de dependencias funcionales, se dirá que f se **deriva** de DF , lo que se representa:

$$DF \vdash f$$

si f se puede obtener por aplicación sucesiva de dichas reglas a partir de DF (o de un subconjunto de DF), es decir, si existe una secuencia de dependencias $f_1, f_2, \dots, f_n$ tal que $f_n = f$, donde cada f_i es bien un elemento de DF o ha sido derivada a partir de las dependencias precedentes aplicando las reglas de derivación.

Vemos, por tanto, que la definición de **consecuencia lógica** se apoya en el cumplimiento de una dependencia en cualquier posible extensión del esquema R , mientras que la definición de **derivación** se refiere a la obtención de nuevas dependencias a partir de la aplicación de unas reglas². Aunque se trata de conceptos distintos, se cumple siempre que si una dependencia f es una **consecuencia lógica** de un conjunto de dependencias, también será posible **derivarla** de dicho conjunto aplicando los axiomas de Armstrong, y viceversa; es decir:

² Estas definiciones de consecuencia lógica y de derivación de un conjunto de dependencias funcionales se puede extender para las dependencias multivaluadas y las dependencias de combinación que veremos en el capítulo siguiente.

$\forall f \mid DF \vdash f \text{ implica } DF \models f$ (corrección)

y

$\forall f \mid DF \models f \text{ implica } DF \vdash f$ (plenitud³)

Una vez enunciados los Axiomas de Armstrong, insistiremos en estas dos propiedades de los mismos.

3.2. Axiomas de Armstrong

Sean los descriptores X, Y, Z, W , los axiomas de Armstrong, es decir, las reglas de derivación de dependencias, establecen:

A1: Reflexividad

Si $Y \subseteq X, X \rightarrow Y$

ya hemos señalado que en este caso la dependencia $X \rightarrow Y$ se dice que es trivial.

A2: Aumentatividad

Si $X \rightarrow Y$ y $Z \subseteq W$, entonces $XW \rightarrow YZ$

A3: Transitividad

Si $X \rightarrow Y$ e $Y \rightarrow Z$, entonces $X \rightarrow Z$

Estos tres axiomas son los básicos; a partir de ellos se pueden deducir otros varios, algunos de los cuales presentamos a continuación.

D1: Proyectividad

Si $X \rightarrow Y$, entonces $X \rightarrow Y'$ si $Y' \subset Y$

D2: Unión a aditividad

Si $X \rightarrow Y$ y $X \rightarrow Z$, entonces $X \rightarrow YZ$

³ Hemos preferido emplear el término "plenitud", ya que "completitud", que se utiliza habitualmente, no existe en castellano.

D3: Pseudotransitividad

Si $X \rightarrow Y$ e $YW \rightarrow Z$, entonces $XW \rightarrow Z$

Ejemplos de aplicación de los axiomas de Armstrong podrían ser los siguientes:

a1) Reflexividad:

$Cód_Estudiante, Nombre_Est \rightarrow Cód_Estudiante$

ya que

$Cód_Estudiante \subset Cód_Estudiante, Nombre_Est$

a2) Aumentatividad:

Si $Cód_Estudiante \rightarrow Nombre_Est$ entonces

$Cód_Estudiante, Cód_Beca, Duración \rightarrow Nombre_Est, Cód_Beca$

ya que

$Cód_Beca \subset Cód_Beca, Duración$

a3) Transitividad

Si $Cód_Estudiante \rightarrow Cód_Beca$ y $Cód_Beca \rightarrow Duración$

entonces

$Cód_Estudiante \rightarrow Duración$

d1) Proyectividad

Si $Cód_Estudiante \rightarrow Nombre_Est, Cód_Beca$, entonces

$Cód_Estudiante \rightarrow Cód_Beca$

ya que

$Cód_Beca \subset Nombre_Est, Cód_Beca$

d2) Unión o aditividad

Si $Cód_Estudiante \rightarrow Nombre_Est$ y $Cód_Estudiante \rightarrow Cód_Beca$

entonces

$Cód_Estudiante \rightarrow Nombre_Est, Cód_Beca$

d3) Pseudotransitividad

Si $Cód_Estudiante \rightarrow DNI$

y

$DNI, Cód_Beca \rightarrow Fecha_Concesión$

entonces

$Cód_Estudiante, Cód_Beca \rightarrow Fecha_Concesión$

Un ejemplo de secuencia de aplicación de los axiomas de Armstrong sería:

Dado el esquema de relación:

$R(A, B, C, D, E; A \rightarrow B, C \rightarrow D, D \rightarrow E)$

Demostrar que $AC \rightarrow ABCDE$

Podemos aplicar los axiomas de Armstrong de la siguiente forma:

- 1) $A \rightarrow B$ (dada)
- 2) $AC \rightarrow ABC$ (aumentatividad de la anterior por AC)
- 3) $C \rightarrow D$ (dada)
- 4) $D \rightarrow E$ (dada)
- 5) $C \rightarrow E$ (transitividad de 3 y 4)
- 6) $C \rightarrow DE$ (unión de 3 y 5)
- 7) $ABC \rightarrow ABCDE$ (aumentatividad de 6 por ABC)
- 8) $AC \rightarrow ABCDE$ (transitividad de 2 y 7)

Luego, AC implica todos los atributos⁴.

Basándonos en los axiomas de Armstrong, podemos dar una nueva definición del concepto de cierre (DF^+) de un conjunto DF de dependencias funcionales:

- 1) DF es un subconjunto de su cierre

$$DF \subseteq DF^+$$

- 2) Toda dependencia $X \rightarrow Y$ derivada de DF mediante la aplicación de los axiomas de Armstrong está en DF^+

$$DF \vdash X \rightarrow Y \in DF^+$$

- 3) Ninguna otra dependencia está en DF^+

Hemos señalado anteriormente que los axiomas de Armstrong constituyen un conjunto correcto y completo de reglas de inferencia para un conjunto DF de dependencias funcionales. Se dice que es un conjunto **correcto** en el sentido de que si una dependencia $X \rightarrow Y$ se ha derivado por aplicación de los axiomas de Armstrong ($DF \vdash X \rightarrow Y$), dicha dependencia es también una consecuencia lógica de DF ($DF \models X \rightarrow Y$) y está contenida en su cierre ($X \rightarrow Y \in DF^+$); es decir, toda dependencia derivada de DF, aplicando los axiomas, se cumple para cualquier extensión r de R.

El conjunto de axiomas es también **completo**, ya que toda dependencia $\{DF^+ - DF\}$ se puede derivar a partir de DF mediante una adecuada aplicación de los axiomas. En otras palabras, aplicando los axiomas de Armstrong a DF podemos encontrar **todas** las dependencias funcionales asociadas a un esquema de relación R (A, DF), es decir, que son una consecuencia lógica del conjunto DF. La demostración de esto la podemos encontrar, además de en ARMSTRONG (1974), en ULLMAN (1988) y en DELOBEL (1982).

Aunque los axiomas de Armstrong facilitan un procedimiento algorítmico para calcular el cierre DF^+ de un conjunto de dependencias, su cálculo consume mucho

⁴ Dos observaciones:

- 1) Hemos demostrado que AC implica todos los atributos, pero no que sea clave, porque no se ha demostrado la condición de minimidad, es decir, no se ha comprobado si:

$$A \longrightarrow ABCDE \text{ y } C \longrightarrow ABCDE$$

Más adelante, al definir formalmente los conceptos de clave y superclave, veremos cómo se determina si un descriptor es clave de una relación.

- 2) La secuencia de derivaciones que se ha aplicado no es la única posible, el lector puede buscar otras secuencias, incluso más cortas, que conduzcan al mismo resultado

tiempo, ya que, aun cuando el número inicial de dependencias sea pequeño, el número total de dependencias en el cierre es muy elevado; para evitar este problema habremos de buscar, en la manipulación de dependencias que exige la teoría de la normalización, procedimientos algorítmicos que no estén basados en el cierre de un conjunto de dependencias. Por otro lado, no todas las dependencias incluidas en el cierre son útiles en el proceso de diseño de una base de datos, por lo cual se introduce el concepto de **recubrimiento** o **cobertura irredundante** también llamado **minimal**.

4. DEFINICIÓN FORMAL DE SUPERCLAVE Y DE CLAVE DE UNA RELACIÓN

El concepto y determinación de las claves candidatas de una relación es muy importante en el proceso de diseño de una base de datos relacional. Aunque en el capítulo anterior se introdujo ya el concepto de clave candidata, podemos ahora definirlo de nuevo más formalmente apoyándonos en las dependencias funcionales, dando asimismo la definición de superclave.

Dado el esquema de relación $R(A, DF)$, se denomina **superclave** SK de la relación R a un subconjunto no vacío de A , tal que $SK \rightarrow A$ es una consecuencia lógica de DF , siendo por tanto, un elemento de su cierre (condición de **unicidad**); expresado formalmente:

$$SK \neq \emptyset \wedge SK \rightarrow A \in DF^+$$

Para el mismo esquema anterior decimos que K es una **clave candidata** de R si, además de ser una superclave, no existe ningún subconjunto estricto K' de K tal que K' implique también a A (condición de **minimidad**); expresado formalmente:

$$K \neq \emptyset \wedge K \rightarrow A \in DF^+ \wedge \nexists K' \subset K \mid K' \rightarrow A$$

la clave es, por tanto, un caso especial de superclave.

En el ejemplo que exponíamos anteriormente de secuencia de aplicación de los axiomas de Armstrong obteníamos un descriptor AC que implicaba a todos los atributos, es decir, era una superclave; sin embargo, no podíamos afirmar que AC fuese una clave porque no demostrábamos la condición de que fuese mínimo, es decir, que no existía ningún subconjunto estricto de AC (esto es A o C) que no implicase a todos los atributos. La única dependencia que tiene a A como implicante es $A \rightarrow B$, y para C como implicante tendríamos $C \rightarrow D, E$, luego:

$$\begin{array}{l} A \not\rightarrow ABCDE \\ C \not\rightarrow ABCDE \end{array}$$

Por tanto, AC es una clave de la relación.

Se denominan atributos principales de una relación a aquellos que forman parte de alguna clave candidata, mientras que el resto se dice que son atributos no principales. Así, en el ejemplo anterior A y C son atributos principales, y B, D, y E son no principales (siempre que se demuestre por aplicación de los axiomas de Armstrong —lo que en este ejemplo no es difícil— que AC es la única clave)

Existen algunos algoritmos que permiten hallar todas las claves presentes en una relación y, por tanto, los atributos principales y no principales de la misma, lo que es fundamental para la determinación de la forma normal en la que se encuentra una relación y para aplicar el proceso de normalización. El cálculo de todas las claves de una relación es algorítmicamente más complejo que comprobar si un descriptor es o no clave de una relación. En el epígrafe 5.5 exponemos la forma de determinar si un descriptor es clave en base a su cierre transitivo.

La comprobación de si un descriptor es clave, así como el cálculo de todas las claves de una relación y, en general, la manipulación algorítmica de las dependencias funcionales, se basa en el concepto de **cierre de un descriptor**. Aunque los algoritmos de cierre de un descriptor (algoritmo de Ullman) y los de determinación de claves, junto con otros relativos al proceso de normalización, se verán en el capítulo 7, seguidamente expondremos los correspondientes conceptos y definiciones.

5. MANIPULACIÓN DE DEPENDENCIAS FUNCIONALES EN BASE AL CIERRE TRANSITIVO DE UN DESCRIPTOR

Si deseamos disponer de métodos algorítmicos eficientes para el diseño de bases de datos relacionales, hemos de abordar adecuadamente una serie de cuestiones relacionadas con la manipulación de dependencias. Los principales problemas que se nos plantean son:

- A) Determinar si una dependencia $X \rightarrow Y$ pertenece al cierre DF^+
- B) Encontrar un procedimiento algorítmico eficiente, por tanto no basado en el cierre de un conjunto de dependencias, para determinar la equivalencia entre dos conjuntos de dependencias.
- C) Hallar un **recubrimiento irredundante**, llamado también **minimal**, lo que es base para abordar el tema de la normalización, tanto en los algoritmos de síntesis como de análisis.
- D) Verificar si un descriptor es clave de un esquema de relación.
- E) Obtener todas las claves candidatas de un esquema relación.

A fin de abordar estos problemas es necesario definir el cierre transitivo de un descriptor.

5.1. Cierre transitivo de un descriptor⁵

Dado un esquema de relación $R(A, DF)$, se define el cierre transitivo de un descriptor X de R respecto al conjunto de dependencias DF , que se denota:

$$X^+_{DF}$$

como un subconjunto de los atributos de A tales que

$$X \rightarrow X^+_{DF} \in DF^+$$

siendo X^+_{DF} máximo en el sentido de que la adición de cualquier atributo vulneraría la condición anterior.

El algoritmo más conocido para el cálculo del cierre de un descriptor es el de Ullman, que presentamos en el capítulo 7, junto con otro de menor complejidad algorítmica. El algoritmo de Ullman comienza haciendo:

$$X^+_{DF} = X$$

y va añadiendo a X todos los atributos que vengan implicados por X obtenidos aplicando el axioma de transitividad al conjunto DF de dependencias.

El cálculo del cierre nos va a permitir determinar si una dependencia $X \rightarrow Y$ se puede derivar de un conjunto de dependencias, así como obtener las claves de un esquema y saber si un descriptor es clave. También el cierre de un descriptor es la base para deducir la equivalencia de dos conjuntos de dependencias y la cobertura minimal de un conjunto de dependencias; es decir, permite dar solución a los problemas que hemos enumerado anteriormente.

Ejemplo:

Dada la relación $R(\{CE, NE, P, G, CP, C\}, DF)$

$$DF = \{CE \rightarrow NE, NE \rightarrow CE, P \rightarrow CE, G \rightarrow P, (CP, P) \rightarrow G, CE \rightarrow C, P \rightarrow C\}$$

hallar el cierre del descriptor (CP, P) .

$$CP, P \rightarrow CP, P$$

$$CP, P \rightarrow CP, P, G$$

$$CP, P \rightarrow CP, P, G, CE$$

⁵ Es preciso no confundir el cierre transitivo de un descriptor con el cierre de un conjunto de dependencias funcionales.

$CP, P \rightarrow CP, P, G, CE, NE$

$CP, P \rightarrow CP, P, G, CE, NE, C$

Luego el cierre transitivo del descriptor es:

$(CP, P)^+ = CP, P, G, CE, NE, C$

5.2. Determinación de si una dependencia está implicada por un conjunto de dependencias (pertenece a su cierre)

Dado un conjunto DF, comprobar si una dependencia $X \rightarrow Y$ pertenece a DF^+ .

Se calcula el cierre X_{DF}^+ de X, si $Y \subseteq X_{DF}^+$ la dependencia $X \rightarrow Y \in DF^+$ (o lo que es igual $DF \models X \rightarrow Y$), en caso contrario $X \rightarrow Y \notin DF^+$.

Ejemplo:

Dado el conjunto de dependencias funcionales anterior, comprobar si la dependencia $NE \rightarrow C$ pertenece al cierre DF^+ .

Se calcula el cierre de NE

$NE^+ = NE, CE, C$

Como C está en el cierre de NE, $NE \rightarrow C$ pertenece a DF^+ .

5.3. Equivalencia de dos conjuntos de dependencias

El problema de la equivalencia de dos conjuntos de dependencias es fundamental en el proceso de normalización, a fin de comprobar si la transformación de un esquema relacional se ha realizado conservando la semántica, al menos en lo que a dependencias se refiere.

Decimos que dos conjuntos de dependencias DF_1 y DF_2 son equivalentes, o que son recubrimientos mutuos, si sus cierres son iguales:

$$DF_1^+ = DF_2^+$$

El problema es, al igual que ocurre en todo procedimiento que lleva asociado el cálculo del cierre de un conjunto de dependencias, el coste computacional de este algoritmo. Para evitar el cálculo de ambos cierres, se puede comprobar, por el procedimiento que acabamos de exponer, si cada dependencia de DF_1 se encuentra en

DF_2 y, viceversa, si cada dependencia de DF_2 se encuentra en DF_1 , lo que lleva a calcular el cierre de todos los implicantes de DF_2 con respecto a DF_1 y de todos los implicantes de DF_1 con respecto a DF_2 .

Si para toda dependencia $X \rightarrow Y$ de DF_2 se cumple

$$a) Y \subseteq X^+_{DF_1}$$

significa que toda dependencia de DF_2 está en DF_1 y, por tanto, DF_1 es un recubrimiento de DF_2 .

Recíprocamente, si para toda dependencia $Z \rightarrow W$ de DF_1 , se cumple

$$b) W \subseteq Z^+_{DF_2}$$

significa que toda dependencia de DF_1 está en DF_2 y, por tanto, DF_2 es un recubrimiento de DF_1 .

Si se cumplen a) y b), DF_1 y DF_2 son mutuamente recubrimientos, luego son equivalentes.

Mediante este procedimiento podemos determinar la equivalencia de dos conjuntos de dependencias sin necesidad de calcular sus cierres, lo que resulta menos costoso computacionalmente.

Ejemplo:

Dados los siguientes conjuntos de dependencias:

$$DF_1 = \{Cód_Curso \rightarrow Nombre, Nombre \rightarrow Cód_Curso, \\ Cód_Curso \rightarrow Cód_Departamento, Cód_Curso \rightarrow Cód_Programa\}$$

$$DF_2 = \{Cód_Curso \rightarrow Nombre, Nombre \rightarrow Cód_Curso, \\ Nombre \rightarrow Cód_Departamento, Nombre \rightarrow Cód_Programa\}$$

Las dependencias $Cód_Curso \rightarrow Nombre$ y $Nombre \rightarrow Cód_Curso$ están en ambos conjuntos, por lo que las únicas dependencias de DF_1 que no están en DF_2 son $Cód_Curso \rightarrow Cód_Departamento$ y $Cód_Curso \rightarrow Cód_Programa$; hemos de calcular, por tanto, el cierre de $Cód_Curso$ con respecto al conjunto DF_2 , que es:

$$Cód_Curso^+_{DF_2} = Cód_Curso, Nombre, Cód_Departamento, Cód_Programa$$

como *Cód_Departamento* es un descriptor contenido en el cierre, al igual que *Cód_Programa*, hemos demostrado que todas las dependencias de DF_1 están en DF_2^+ , luego DF_2 es un recubrimiento de DF_1 .

Análogamente, demostraríamos que las dependencias de DF_2 : *Nombre* $\rightarrow$ *Cód_Departamento* y *Nombre* $\rightarrow$ *Cód_Programa* están contenidas en DF_1^+ , sólo con hallar el cierre de *Nombre* con respecto a DF_1 , y comprobaríamos que *Cód_Departamento* y *Cód_Programa* están en dicho cierre, por lo que DF_1 es un recubrimiento de DF_2 . Por tanto, DF_1 y DF_2 son equivalentes.

5.4. Recubrimiento irredundante de un conjunto de dependencias

Un conjunto de dependencias funcionales se dice que es **mínimo** cuando cumple:

- Todas sus dependencias son elementales⁶
- No existe en el conjunto de dependencias ninguna redundante, es decir, que se pueda deducir del resto aplicando los axiomas de Armstrong

De todos los posibles conjuntos equivalentes a un conjunto dado de dependencias, hay algunos de ellos que son mínimos, por lo que se dice que son recubrimientos **irredundantes** (también llamados minimales) del conjunto dado de dependencias.

Resulta interesante, puesto que las dependencias son restricciones semánticas, eliminar todas aquellas que sean redundantes, a fin de reducir el coste de mantenimiento de la integridad de la base de datos, sin, en principio, perjudicar la semántica⁷, ya que estamos sustituyendo el conjunto de dependencias de partida por un conjunto equivalente además de irredundante. Por esta razón, además de a efectos de reducir la complejidad algorítmica, los algoritmos de normalización y los de cálculo de claves candidatas parten siempre de recubrimientos irredundantes.

Podemos definir un **recubrimiento irredundante** de un conjunto de dependencias funcionales asociadas a un conjunto de atributos A , a un subconjunto no estricto de las dependencias elementales del conjunto inicial DF , tal que cumpla:

- Ninguna de las dependencias funcionales elementales en DF es redundante, es decir, si se elimina cualquiera de las dependencias de DF , el nuevo conjunto de dependencias DF' no es equivalente a DF (no tiene el mismo cierre).
- Todas las dependencias funcionales entre los atributos A están en DF^+ .

⁶ Recuérdese la definición de dependencia elemental, que es aquella que tiene un *único implicado*, siendo además **plena** y **no trivial**. También dijimos que son las únicas interesantes en la teoría de la normalización.

⁷ Decimos *en principio*, pues, como veremos posteriormente, esto quizá no sea rigurosamente cierto, ya que puede ocurrir que aunque conceptualmente la existencia de dos conjuntos de dependencias equivalentes significa que toda la semántica del uno está contenida en el otro, y viceversa, desde el punto de vista del usuario puede no ser lo mismo, ya que ciertas dependencias, al igual que algunas relaciones, quizá no sean significativas para los usuarios.

La definición de recubrimiento irredundante se basa en los conceptos de **atributo extraño**⁸, y de dependencia redundante.

Atributo extraño Dada la dependencia $X \rightarrow Y$ de DF, un atributo A perteneciente a X se dice que es un atributo extraño en la dependencia si la dependencia $(X - A) \rightarrow Y$ se encuentra en DF^+ .

Dependencia redundante Una dependencia funcional f de DF se dice que es redundante si puede derivarse de $\{DF - f\}$ mediante la aplicación de los axiomas de Armstrong.

Una definición matemática de recubrimiento irredundante dada por ATKINS (1988) es: Dado un conjunto M se dice que es recubrimiento irredundante si:

- Todas sus dependencias son elementales
- No hay atributos extraños, es decir, no existe $X \rightarrow A$ en M , con Z incluido en X ($Z \subset X$), tal que M esté contenido en el cierre de

$$M - (X \rightarrow A) \cup (Z \rightarrow A)$$

- No existen dependencias redundantes, es decir, no existe $X \rightarrow A$ en M tal que

$$M - (X \rightarrow A) \text{ es equivalente a } M$$

Dado un conjunto de dependencias DF siempre es posible encontrar un recubrimiento irredundante.

Jyrky Nummenmaa en 1990 publicó una nota en la que corrige la definición de recubrimiento irredundante dada por Atkins, demostrando con el siguiente ejemplo que la condición impuesta para la eliminación de atributos extraños es necesaria, pero no suficiente.

Ejemplo:

Sea el esquema de relación

$$R(\{A, B, C\}, \{AB \rightarrow C\})$$

para la eliminación de atributos extraños podemos sustituir $AB \rightarrow C$ por $A \rightarrow C$ y se cumple la condición de que $AB \rightarrow C$ está contenido en el cierre de $(A \rightarrow C)^+$, por el

⁸ Conservamos esta denominación de atributo extraño porque es el más extendido, pero consideramos que la expresión no es afortunada y habría sido mejor llamarlo atributo *ajeno* a una dependencia en un conjunto de dependencias.

axioma de aumentatividad de Armstrong, obteniendo que el recubrimiento irredundante del esquema es $(A \rightarrow C)$, lo cual es erróneo, ya que no se mantiene la propiedad de equivalencia de conjuntos de dependencias.

La solución dada en NUMMENMAA (1990) es que, para la eliminación de atributos extraños, además de comprobar la condición impuesta anteriormente, se ha de comprobar también que no existe $Z \subset X$ tal que

$$M - (X \rightarrow A) \cup (Z \rightarrow A)$$

esté contenido en el cierre de M .

El cálculo del recubrimiento minimal se basa en el cierre transitivo de un descriptor, como se puede comprobar en los algoritmos que se incluyen en el capítulo 7.

Veamos a continuación algunos ejemplos de recubrimientos irredundantes de un conjunto de dependencias funcionales.

Sea la relación

ESTUDIANTE ($\{Cód_Estudiante, Dni, Cód_Curso, Cód_Programa\}$
 $\{Cód_Estudiante \rightarrow Dni, Dni \rightarrow Cód_Estudiante,$
 $Dni \rightarrow Cód_Curso, Cód_Programa,$
 $Cód_Estudiante \rightarrow Cód_Curso,$
 $Cód_Curso \rightarrow Cód_Programa\}$)

Los conjuntos de dependencias

$DF_1 = \{ Cód_Estudiante \rightarrow Dni, Dni \rightarrow Cód_Estudiante,$
 $Cód_Estudiante \rightarrow Cód_Curso, Cód_Curso \rightarrow Cód_Programa \}$

$DF_2 = \{ Cód_Estudiante \rightarrow Dni, Dni \rightarrow Cód_Estudiante,$
 $Dni \rightarrow Cód_Curso, Cód_Curso \rightarrow Cód_Programa \}$

son recubrimientos irredundantes de las dependencias funcionales presentes en la relación, mientras que el conjunto original no era irredundante, ni tampoco lo es el siguiente:

$DF_3 = \{ Cód_Estudiante \rightarrow Dni, Dni \rightarrow Cód_Estudiante,$
 $Dni \rightarrow Cód_Programa, Dni \rightarrow Cód_Curso,$
 $Cód_Curso \rightarrow Cód_Programa \}$

ya que $Dni \rightarrow Cód_Programa$ es una dependencia redundante que puede ser deducida a partir de

$$Dni \rightarrow Cód_Curso \text{ y } Cód_Curso \rightarrow Cód_Programa$$

Tampoco es un recubrimiento irredundante:

$$DF_4 = \{ Cód_Estudiante \rightarrow Dni, Dni \rightarrow Cód_Curso, \\ Cód_Curso \rightarrow Cód_Programa \}$$

ya que la dependencia $Dni \rightarrow Cód_Estudiante$ falta y no puede ser deducida de las demás.

De los ejemplos anteriores se deduce que pueden existir varios recubrimientos irredundantes de un mismo conjunto de dependencias, luego dicho recubrimiento **no es único**.

Incluso pueden obtenerse recubrimientos irredundantes cuyo número de dependencias sea distinto. Por ello, podría distinguirse entre recubrimiento *irredundante* o *minimal* y recubrimiento *mínimo*; el primero corresponde al concepto ya definido, mientras que el segundo sería un recubrimiento irredundante cuyo número de dependencias fuese mínimo.

Existen conjuntos irredundantes que no son **mínimos** en cuanto a número de dependencias, y también puede ocurrir que el número de atributos involucrados en cada conjunto de dependencias sea distinto.

Como la idea que subyace en los recubrimientos irredundantes es no sólo reducir la complejidad algorítmica (al disminuir el número de dependencias de partida), sino también, como ya hemos señalado, minimizar el número de restricciones de integridad que han de ser mantenidas en la base de datos, habría de ser un objetivo de diseño conseguir que el número de dependencias, así como el de atributos involucrados en las mismas, sea mínimo, lo que, por otra parte, puede llevar a obtener un mínimo de esquemas relacionales.

Existe, sin embargo, otro objetivo de diseño, en nuestra opinión aún más importante, como es que las dependencias resultantes tengan un significado claro para los usuarios, y éste es un problema que no puede ser resuelto con la teoría de la normalización que realiza transformaciones algorítmicas de tipo sintáctico que pueden conducir a dependencias y a esquemas de relación **absurdos** desde el punto de vista del usuario. En KAMBAYASHI (1988) se pone el siguiente ejemplo:

Dadas las ocho dependencias

Profesor $\leftrightarrow$ *Materia*

Materia $\rightarrow$ *Aula*

Profesor $\leftrightarrow$ *Despacho*

Profesor $\rightarrow$ *Edad*

Profesor $\leftrightarrow$ *Dirección*

que constituyen un recubrimiento irredundante, se pueden obtener otros recubrimientos irredundantes:

RECUBRIMIENTO 1:

Profesor $\rightarrow$ *Materia, Edad*

Materia $\rightarrow$ *Aula, Dirección*

Dirección $\rightarrow$ *Despacho*

Despacho $\rightarrow$ *Profesor*

RECUBRIMIENTO 2:

Profesor $\rightarrow$ *Dirección*

Materia $\rightarrow$ *Despacho*

Dirección $\rightarrow$ *Edad, Materia*

Despacho $\rightarrow$ *Aula, Profesor*

Los tres conjuntos de dependencias son recubrimientos irredundantes (podrían haberse obtenido otros), pero mientras que el primero contiene **ocho** dependencias, los otros dos sólo tienen **seis** cada uno. Luego, existen conjuntos irredundantes que no son **mínimos** en cuanto a número de dependencias. Aunque éste no es el caso, también podría ocurrir que el número de atributos involucrados en cada conjunto de dependencias fuese distinto.

Sin embargo, mientras que el conjunto de dependencias inicial es conceptualmente muy claro, el recubrimiento 1 lo es mucho menos, y el 2 produciría bastante estupor a los usuarios, que no podrían comprender la razón por la cual la dirección determine la edad de un profesor y la materia que imparte⁹.

5.5. Determinación de si un descriptor es clave de una relación

Otro de los problemas que se plantea en relación con la manipulación de dependencias es cómo determinar si un descriptor es o no clave de una relación.

⁹ En el modelo E/R no se producen estos problemas, dado que tanto las entidades como las interrelaciones reflejan directamente la concepción que del universo del discurso tiene el usuario.

Dada una relación $R(A, DF)$, se trata de comprobar si un descriptor X es una clave.

Se calcula el cierre X^+_{DF} de X si

$X^+_{DF} \rightarrow A$, X es una superclave

$X^+_{DF} \not\rightarrow A$, X no es clave

Si X es una superclave, se hallan todos los subconjuntos X'_i del descriptor, si algún

$X'^+_{i\ DF} \rightarrow A$, X no es clave

En caso contrario

X es **una** clave candidata.

Ejemplo:

Dado el esquema de relación siguiente:

SE_MATRICULA (Cód_Estudiante, Nombre, Cód_Curso, Fecha, Cód_Programa, Cód_Departamento)

donde se dan las siguientes dependencias funcionales:

Cód_Estudiante $\rightarrow$ Nombre

Nombre $\rightarrow$ Cód_Estudiante

Cód_Curso $\rightarrow$ Cód_Programa

Cód_Programa $\rightarrow$ Cód_Departamento

Cód_Estudiante, Cód_Curso $\rightarrow$ Fecha

para determinar si el descriptor (*Cód_Estudiante, Cód_Curso*) es una clave, calculamos el cierre del mismo respecto al conjunto de dependencias anterior, obteniendo:

(Cód_Estudiante, Cód_Curso, Nombre, Cód_Programa, Fecha, Cód_Departamento)

Como el cierre coincide con el conjunto de atributos de la relación, podemos afirmar que el descriptor (*Cód_Estudiante, Cód_Curso*) es una superclave. Además como el cierre transitivo del descriptor *Cód_Estudiante* es (*Cód_Estudiante, Nombre*) y el de *Cód_Curso* es (*Cód_Curso, Cód_Programa, Cód_Departamento*), ninguno de

los cuales coincide con el conjunto de todos los atributos de la relación, podemos decir que el descriptor (*Cód_Estudiente, Nombre*) es una clave.

Otra clave la constituye el descriptor (*Nombre, Cód_Curso*).

5.6. Obtención de las claves candidatas de un esquema

El último de los problemas que nos planteábamos en relación con la manipulación de dependencias era la obtención de todas las claves candidatas de un esquema; se trata de un proceso algorítmico complejo que se basa también en el cierre de un descriptor y que parte de un recubrimiento irredundante del conjunto inicial de dependencias.

En los esquemas que representan objetos del mundo real, como los obtenidos a partir de un esquema conceptual en el ME/R, es muy fácil determinar las claves; sin embargo, en la relación universal o en ejemplos de laboratorio con gran cantidad de dependencias y con atributos muchas veces solapados, el problema de determinación de todas las claves candidatas se complica considerablemente, siendo preciso disponer de algoritmos para ello.

DELOBEL (1982) expone un algoritmo en el marco del álgebra de Boole que genera todas las claves de un esquema. En el capítulo 7 expondremos otro algoritmo de determinación de claves.

En anexo a este capítulo presentamos un procedimiento que permite obtener todas las claves candidatas de un esquema de relación. Se trata de un procedimiento que resulta muy sencillo de aplicar en la mayor parte de los casos, aunque puede complicarse si existen muchas dependencias con atributos solapados.

ANEXO: PROCEDIMIENTO DE CÁLCULO DE CLAVES

Presentamos a continuación un procedimiento para el cálculo de las claves de una relación, ilustrándolo con algunos ejemplos.

Sea al esquema de relación $R(A, DF^{10})$

Se eliminan de DF, tal como se verá posteriormente, todas aquellas dependencias que supongan la equivalencia de descriptores, dejando sólo uno de cada grupo de descriptores equivalentes.

Si ya no existen descriptores equivalentes se ha de tener en cuenta lo siguiente:

¹⁰ Suponemos que DF es un recubrimiento irredundante.

- a) Todo atributo independiente (que no interviene en ninguna dependencia funcional ni como implicante ni como implicado) forma parte de todas las claves.
- b) Los descriptores equivalentes dan lugar a varias claves.
- c) Ningún atributo implicado que no es implicante forma parte de ninguna clave.
- d) Todo atributo implicante pero no implicado forma parte de todas las claves (siempre que no tenga otros equivalentes).
- e) Aquellos atributos que son implicantes e implicados pueden formar parte de alguna clave.

La forma de actuar es la siguiente:

Paso 1: Eliminación de descriptores independientes

Se eliminan de la relación todos los atributos que no entren a formar parte de ninguna dependencia, obteniendo una relación R_{s_i}

Ejemplo 1:

Sea el esquema de relación:

$$R(\{A,B,C,D,E,F,G,H,I,J\}; \{AB \rightarrow C, C \rightarrow AB, E \rightarrow D, D \rightarrow E, E \rightarrow F, F \rightarrow E, ABD \rightarrow G, CF \rightarrow H\})$$

Los atributos I, J son independientes porque no forman parte de ninguna dependencia funcional, luego, en este primer paso, se eliminan de la relación.

Paso 2: Eliminación de descriptores equivalentes

Siempre que existan dependencias de la forma $X \rightarrow Y$ e $Y \rightarrow X$, se dice que X e Y son equivalentes, lo que se puede representar por $X \leftrightarrow Y$. Puede ocurrir que sean más de dos descriptores los que son equivalentes:

$$X \leftrightarrow Y \leftrightarrow Z \text{ (X,Y,Z es un grupo de tres descriptores equivalentes).}$$

Por cada grupo de descriptores equivalentes, se elige uno (por ejemplo X), eliminando las cuatro dependencias anteriores de DF y se sustituyen en las dependencias restantes los descriptores eliminados (por ejemplo Y,Z) por el atributo que se ha elegido del grupo (X en este caso). Esto se hace para cada grupo de descriptores equivalentes.

Ejemplo 2:

Sea el esquema de relación del ejemplo 1, una vez eliminados los atributos independientes:

$$R_{si} (\{A, B, C, D, E, F, G, H\}, \{AB \leftrightarrow C, D \leftrightarrow E \leftrightarrow F, ABD \rightarrow G, CF \rightarrow H\})$$

Existen dos grupos de descriptores equivalentes:

- a) AB y C
- b) D, E y F

Del grupo a) nos quedaríamos, por ejemplo, con C y del grupo b) con D (eliminaríamos, por tanto, AB^{11} , E y F); la relación resultante sin equivalencias sería:

$$R_{sie} (\{C, D, G, H\}; \{CD \rightarrow G, CD \rightarrow H\})$$

Hay que tener en cuenta que, a veces, las dependencias de equivalencia no se dan directamente sino que aparecen en un ciclo:

Ejemplo 3:

$$\begin{aligned} AD &\rightarrow B, \\ B &\rightarrow C \\ C &\rightarrow AD \end{aligned}$$

Lo que implica:

$$AD \rightarrow C \text{ y } C \rightarrow AD$$

es decir:

C y AD son equivalentes

También:

$$B \rightarrow AD \text{ y } AD \rightarrow B$$

Luego: B y AD son equivalentes

$$\text{Por tanto: } AD \leftrightarrow B \leftrightarrow C$$

¹¹ AB como descriptor, no cada uno de los atributos A y B por separado; si éstos entran en otras relaciones por separado no se eliminan.

La R_{se} podría ser cualquiera de estas tres:

$$R1_{sie} = (AD; \emptyset)$$

$$R2_{sie} = (B; \emptyset)$$

$$R3_{sie} = (C; \emptyset)$$

Nos quedaríamos, por ejemplo, con la primera.

Cuando, como resultado de este paso, las relaciones no tienen dependencias, los atributos de las mismas son independientes; así, en $R1_{sie}$ A y D son atributos independientes.

Paso 3: *Determinación de un descriptor (en el que no haya implicados) que sea clave de R_{sie} .*

Basándonos en que todos los atributos de una relación R_{sie} , sin atributos independientes ni equivalencias, que son implicantes pero no implicados forman parte de la clave, tomamos estos atributos y con ellos formamos una clave posible (K_p) si no hay ningún otro implicante que, a la vez, sea implicado K_p , es una clave.

En caso contrario se halla el cierre de la clave posible K_p^+ , y si en el cierre aparecen todos los atributos de R_{sie} , la clave posible es una clave (el resto de los atributos de R_{sie} son implicados y, por tanto, no son parte de la clave).

Una vez obtenida la clave de R_{sie} nos vamos al paso 5.

Ejemplo 4:

En la R_{sie} del ejemplo 2, CD es el único implicante, pero no implicado, luego una K_p sería CD, como el resto son sólo implicados, CD es clave de R_{sie} ¹² (no haría falta hallar el cierre de CD).

Puede ocurrir que en el paso 2 no obtengamos ninguna clave bien porque:

- El conjunto de DF es vacío como en el ejemplo 3, en cuyo caso vamos al paso 5 ya que todos los atributos son independientes¹³.
- En el cierre de la K_p no aparecen todos los atributos R_{sie} , por tanto la clave posible no es una clave, en cuyo caso vamos al paso 4.

¹² Las claves de R las hallaremos posteriormente.

¹³ Nunca puede ocurrir que no exista ningún implicante que no sea implicado, porque ello significaría la existencia de atributos equivalentes, lo que no es posible porque éstos han sido ya eliminados (recuérdese que estamos trabajando con R_{sie}). Sí puede ocurrir, como en el ejemplo 3, que al tratar los descriptores equivalentes, el conjunto de dependencias resultantes sea vacío, de forma que atributos que antes no eran independientes lo sean ahora.

Ejemplo 5:

Sea $R_{\text{sie}} (\{A,B,C,D,E,F\}; AB \rightarrow C, DE \rightarrow F, F \rightarrow D)$

$$K_p = ABE;$$

$$K_p^+ = ABCE$$

luego K_p no es clave, por lo que iríamos al paso 4

Paso 4: *Determinación de un descriptor clave de R_{sie} (en el que puede haber implicados siempre que sean también implicantes).*

Si es posible se obtiene una partición eliminando de R_{sie} todos aquellos atributos que entran en K_p^+ y que no forman parte de otras dependencias funcionales, distintas a las que han servido para calcular K_p^+ . Obtenemos así una nueva relación R'_{sie} .

Ejemplo 6:

La R_{sie} del ejemplo 5 tenía una K_p cuyo cierre era:

$$K_p^+ = ABCE$$

Obtenemos una nueva relación R'_{sie} eliminando de R_{sie} los atributos A B C que forman la primera dependencia funcional (no eliminamos E porque en la dependencia de la que forma parte aparecen D y F que no están en R_{sie}) y nos queda:

$$R'_{\text{sie}} (\{DEF\}; \{DE \rightarrow F, F \rightarrow D\})$$

En la partición resultante se obtiene una clave provisional K'_p con los implicantes que estaban también en K_p (en este caso sólo E) añadiendo un nuevo implicante que, a su vez, es también implicado (por ejemplo F); se halla el cierre K'^{+}_p , si éste contiene todos los atributos de R'_{sie} es una clave, en caso contrario se añade un nuevo descriptor hasta obtener una clave. Se repite esta operación porque pueda haber más claves.

Una vez obtenidas las claves de la partición se une cada una de ellas con la clave obtenida en el paso 3 para obtener así las claves de R_{sie} .

En la anterior partición R'_{sie} , procederíamos de la siguiente forma a fin de obtener sus claves:

Formaríamos una clave provisional K'_p con E que es sólo implicante (y, por tanto, está en R_{sie}), añadiendo un descriptor implicante e implicado, por ejemplo F:

$$K'_p = EF$$

$$K'^+_p = EFD$$

Luego EF es una clave de la partición R'_{sie}

Otra clave sería ED

Las claves de R_{sie} serían:

ABEF

ABED

Si no fuese posible obtener la partición R'_{sie} se actuaría de la misma manera que se acaba de explicar, pero con R_{sie} .

Paso 5: *Tratamiento de atributos independientes para obtener una clave de la relación original*

A las claves de R_{sie} obtenidas en el paso 3 o en el paso 4 se añaden los atributos independientes obtenidos en el paso 1 ó 2.

Ejemplo 7:

La relación del ejemplo 1 tenía dos atributos independientes I y J. Una vez eliminados, en el ejemplo 2 hemos hallado la correspondiente relación R_{sie} (sin equivalencias ni atributos independientes), y en el ejemplo 4 hemos hallado la clave CD de R_{sie} . Si a CD le añadimos los atributos independientes I, J tenemos CDIJ que es la clave de R.

Ejemplo 8:

En la relación del ejemplo 3, después de eliminar las equivalencias, obteníamos, AD como descriptor independiente en R_{sie} , ya que no existen dependencias. Por tanto, una clave de la relación sería AD.

Paso 6: *Tratamiento de descriptores equivalentes*

Cuando en el paso 2 se han obtenido descriptores equivalentes habrá que obtener todas las claves, sustituyendo, en las claves obtenidas en el paso 5 (si hubiese atributos

independientes) o en los pasos 3 ó 4 (si no los hubiera), los descriptores por sus equivalentes; de esta forma se obtiene todas las claves candidatas.

Ejemplo 9:

La relación del ejemplo 1, en la cual existían las siguientes equivalencias (obtenidas en el ejemplo 2):

$$AB \leftrightarrow C \text{ y } D \leftrightarrow E \leftrightarrow F$$

Tiene la clave CDIJ (obtenida en el ejemplo 7). El tratamiento de descriptores equivalentes nos daría las 6 claves siguientes:

CDIJ (ya obtenida en el paso 5, ejemplo 7)

CEIJ

CFIJ

ABDIJ

ABEIJ

ABFIJ

Ejemplo 10:

La relación del ejemplo 3, en la cual existían las siguientes equivalencias:

$AD \leftrightarrow B \leftrightarrow C$, tendría las tres claves siguientes

AD (ya obtenida en el paso 5)

B

C

TEORIA DE LA NORMALIZACION: FORMAS NORMALES BASADAS EN LAS DEPENDENCIAS FUNCIONALES

Los modelos de datos son instrumentos que nos ayudan a representar la realidad, es decir, nuestro universo del discurso (UD). El proceso de diseño de una base de datos consiste en representar un determinado UD mediante los objetos que proporciona el modelo de datos que estamos utilizando, aplicando para ello las reglas de dicho modelo, como prohibición de un determinado tipo de asociaciones o posibilidad de incluir ciertas restricciones. Cuando se diseña una base de datos mediante el modelo relacional, al igual que ocurre en otros modelos de datos, tenemos distintas opciones, es decir, podemos obtener diferentes esquemas relacionales, y no todos ellos son equivalentes, ya que unos van a representar la realidad mejor que otros.

Con la teoría de la normalización, que introducimos en este capítulo, se consigue una formalización en el diseño lógico de bases de datos relacionales, lo que permite disponer de instrumentos algorítmicos¹ de ayuda al diseño y, de esta forma, poder desarrollar programas que automaticen el diseño en el modelo relacional.

En este capítulo vamos a ver qué propiedades debe tener un esquema relacional para representar adecuadamente la realidad y cuáles son los problemas que se pueden derivar de un diseño inadecuado. Expondremos la teoría de la normalización, que permite afrontar el problema de diseño de bases de datos relacionales de una manera rigurosa y objetiva, estudiando la forma de llevar a cabo la normalización de un esquema relacional, tanto por el método de análisis como por el de síntesis, para lo

¹ El capítulo 7 está dedicado a presentar los procedimientos algorítmicos relativos a la teoría de la normalización.

cual nos basamos en el concepto de dependencia funcional que hemos definido en el capítulo anterior.

1. NECESIDAD DE UN MÉTODO FORMAL DE DISEÑO RELACIONAL

Como veíamos al estudiar la estructura del modelo relacional, la información de nuestra base de datos puede representarse por medio de un conjunto de objetos (**relaciones** y **dominios**) y de un conjunto de reglas de integridad.

En el modelo relacional, como en los demás modelos de datos, el diseño de una base de datos se puede abordar de dos formas distintas:

- A) Obteniendo el esquema relacional directamente a partir de la observación de nuestro universo del discurso, de forma que plasmemos nuestra percepción del mismo en un conjunto de esquemas de relación, los cuales contendrán los atributos y las restricciones de integridad que representan los objetos y reglas que hemos podido captar en nuestro análisis del mundo real.
- B) Realizando el proceso de diseño en dos fases, en la primera se lleva a cabo el diseño conceptual, por ejemplo en el modelo E/R, obteniéndose el correspondiente esquema conceptual; en la segunda, éste se transforma en un esquema relacional, siguiendo unas determinadas reglas de transformación².

Estas relaciones que resultan de la observación del mundo real o de la transformación al modelo relacional del esquema E/R elaborado en la etapa de modelado conceptual, pueden presentar algunos problemas, derivados de fallos en la percepción del UD, en el diseño del esquema E/R, o en el paso al modelo relacional; entre estos problemas cabe destacar los siguientes:

- Incapacidad para almacenar ciertos hechos.
- Redundancias y, por tanto, posibilidad de inconsistencias.
- Ambigüedades.
- Pérdida de información (aparición de *tuplas espurias*).
- Pérdida de dependencias funcionales, es decir, de ciertas restricciones de integridad que dan lugar a interdependencias entre los datos.
- Existencia de valores nulos (inaplicables)
- Aparición, en la base de datos, de estados que no son válidos en el mundo real (anomalías de inserción, borrado y modificación).

² Más adelante se exponen con mayor profundidad y rigor las reglas de transformación del modelo E/R al modelo relacional.

En definitiva, el esquema relacional debe ser siempre analizado para comprobar que no presenta los problemas anteriormente citados, evitando la pérdida de información y la aparición de estados que no son válidos en el mundo real.

Veamos un ejemplo: en la figura 5.1. se muestra una relación denominada ESTUDIANTE_SOLICITA_BECA que almacena datos sobre los estudiantes que solicitan becas y sobre las propias becas además de la fecha en la que un estudiante solicita una beca. Si observamos con atención esta relación, vemos que presenta varios de los problemas enumerados anteriormente:

ESTUDIANTE_SOLICITA_BECA								
Cód_Estud	Nombre_E	Apellido	DNI	Dirección	Cód_Beca	Nombre	Requisito	Fecha
012323	Roberto	Hens	456367	Antonio López 43	A22321	METRICA	Ing. Téc.	10/10/98
763476	Luis	García	345347	Av. Ciudades 29	B56784	ERASMU	Ing. Téc.	12/11/98
763476	Luis	García	345347	Av. Ciudades 29	A22321	METRICA	Ing. Téc.	14/10/98
763476	Luis	García	345347	Av. Ciudades 29	G65434	HIMMPA	Ingenie.	15/09/98
012323	Roberto	Hens	456367	Antonio López 43	G65434	HIMMPA	Ingenie.	17/09/98
987765	Gregorio	Celada	885764	Pl. Países 67	G65434	HIMMPA	Ingenie.	21/09/98
012323	Roberto	Hens	456367	Antonio López 43	B56784	ERASMU	Ing. Téc.	11/11/98
987765	Gregorio	Celada	885764	Pl. Países 67	B56784	ERASMU	Ing. Téc.	10/10/98
012323	Roberto	Hens	456367	Antonio López 43	A22321	METRICA	Ing. Téc.	12/11/98
232457	Mercedes	García	809234	Río Miño 2	A22321	METRICA	Ing. Téc.	17/09/98
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figura 5.1. Ejemplo de diseño inadecuado

- Gran cantidad de redundancia; ya que el nombre, apellidos, DNI y dirección de los estudiantes se repite por cada beca que hayan solicitado, y algo análogo sucede, cuando una beca la solicita más de un estudiante, con el nombre de la beca, y los requisitos.
- Anomalías de modificación; ya que podemos, por ejemplo, cambiar la dirección de un estudiante en una tupla y por error no modificarla en el resto de las que corresponden al mismo estudiante, lo que da lugar a inconsistencias.

- Anomalías de inserción; ya que si se quisiera incluir información sobre algún estudiante que no hubiera solicitado ninguna beca, no sería posible, al formar el atributo `Cod_Beca` parte de la clave primaria de la relación; ni tampoco podríamos introducir becas no solicitadas (recuerde el lector la regla de integridad de entidad que no permite nulos en ningún atributo que forme parte de una clave primaria). Además, la inserción de una beca que solicitase más de un estudiante obligaría a incluir varias tuplas en la base de datos.
- Anomalías de borrado; ya que si quisiéramos dar de baja una beca, también se perderían datos sobre los estudiantes que la solicitan (si éstos hubieran solicitado sólo esa beca) y, viceversa, si borramos un estudiante desaparecerían de nuestra base de datos las becas solicitadas por él (a no ser que hubieran sido solicitadas por otros estudiantes).

Vemos, por tanto, que la actualización (alta, baja o modificación) de una sola beca o de un solo estudiante nos puede obligar a actualizar más de una tupla, dejándose la integridad de la base de datos en manos del usuario; además de la falta de eficiencia asociada a estas múltiples actualizaciones.

Si se hubiera llevado a cabo un diseño riguroso, no se habría presentado una relación de este tipo. El problema es que a menudo no se llega a comprender completamente el UD, debido a una excesiva premura al realizar el análisis, o bien por carecer el analista de conocimientos sobre metodologías de diseño de bases de datos o de experiencia para aplicarlas adecuadamente; también los problemas pueden provenir del usuario que no conoce bien el universo del discurso o no sabe exponerlo con precisión.

Si se siguiera la metodología de diseño que proponemos mas adelante, realizando un adecuado diseño conceptual en el modelo E/R, seguido de una cuidadosa transformación al modelo relacional, se evitarían estos problemas y se obtendría un esquema relacional exento de errores. Sin embargo, ante las posibles dudas respecto a si un determinado esquema relacional es correcto, será preferible aplicar a dicho esquema un método formal de análisis que determine lo que pueda estar equivocado en el mismo y nos permita llegar a otro que nos asegure el cumplimiento de ciertos requisitos; este método formal es la teoría de la normalización.

2. TEORÍA FORMAL DE LA NORMALIZACIÓN DE ESQUEMAS RELACIONALES

Formalmente plantearemos la teoría de la normalización en los siguientes términos: Dado un conjunto A de atributos y el conjunto D de dependencias existentes entre ellos, que puede considerarse que constituyen un esquema de relación $R(A, D)$ (esquema origen), se trata de transformar, por medio de sucesivas proyecciones, este esquema de partida en un conjunto de n esquemas de relación (esquemas resultantes):

$$\{R_i(A_i, D_i)\}_{i=1}^n$$

tales que cumplan unas determinadas condiciones.

Es evidente que una base de datos no puede estar constituida por una única relación con todos los atributos (la relación **universal**), ya que ello daría lugar a una enorme cantidad de redundancias, provocando las anomalías de actualización a las que antes nos hemos referido. Parece lógico reemplazar el esquema que contiene todos los atributos y todas las dependencias por un conjunto R_i de esquemas equivalentes que cumplan unas ciertas propiedades, a fin de conseguir la mejor representación de nuestro universo del discurso.

Se trata, por tanto, de buscar un conjunto de esquemas R_i que sean **equivalentes** a R (para lo cual será preciso definir el concepto de equivalencia de esquemas) y que sean también **mejores** (en el sentido que enunciaremos) que el esquema de partida.

Son tres las propiedades que han de cumplir los esquemas de relación R_i para ser **equivalentes** a R y **mejores** que R ; estas propiedades son:

- a) Conservación de la información.
- b) Conservación de las dependencias³.
- c) Mínima redundancia de los datos (normalización de las relaciones⁴).

Si se cumplen las dos primeras propiedades, es decir, la transformación de R en $\{R_i\}$ se hace sin pérdida de información ni de dependencias, se dice que:

$\{R_i\}$ es **equivalente** a R

y si las relaciones del esquema relacional resultante $\{R_i\}$ están en formas normales más avanzadas que el esquema origen R , se dice que en este sentido de nivel de normalización y, por tanto, de eliminación de redundancias de datos:

$\{R_i\}$ es “**mejor**” que R

Obsérvese que decimos que es **mejor** en cuanto a su nivel de normalización, pero existen otros criterios para calificar la **bondad** de un esquema relacional, entre otros, su eficiencia frente a las consultas y el de captar mejor la semántica del mundo real.

KAMBAYASHI (1982) expone, además de los anteriores, otros objetivos que debe cumplir el conjunto de relaciones resultantes para conseguir un buen diseño:

³ En este capítulo sólo nos referiremos a las dependencias funcionales.

⁴ En este capítulo sólo trataremos las formas normales basadas en dependencias funcionales, es decir, las tres primeras formas normales y la de Boyce y Codd.

- a) **Minimización de dependencias**, que incluye no sólo minimizar el número de dependencias, sino también el número de atributos contenidos en ellas.
- b) **Minimización de los esquemas resultantes**, que, al igual que el caso anterior, incluye no sólo minimizar su número, sino también el número de sus atributos.

Ya hemos aludido anteriormente a que se pueden obtener recubrimientos irredundantes con distinto número de dependencias partiendo de un mismo conjunto inicial, pero lo que no siempre resulta posible es conseguir que se cumplan a la vez todos estos objetivos.

Expondremos a continuación en qué consiste cada una de las tres propiedades, **conservación de la información**, **conservación de las dependencias** y **normalización de las relaciones**.

2.1. Conservación de la información

La finalidad de esta propiedad es conseguir que el proceso de normalización se lleve a cabo sin pérdida de la información existente en la base de datos; es decir, la información contenida en la relación origen debe ser la misma que la contenida en el conjunto R_i de esquemas resultantes, es lo que a veces se llama *equivalencia de datos*.

Para que se cumpla esta propiedad son precisas dos condiciones:

1) Conservación de los atributos

El conjunto de atributos de los esquemas resultantes ha de ser igual al conjunto de atributos del esquema origen, lo que podemos expresar:

$$\bigcup_{i=1}^n A_i = A$$

Siempre han de existir atributos comunes, es decir que se encuentren en más de una relación, pero la unión de todos los atributos de las relaciones resultantes debe ser igual a los atributos de la relación origen.

2) Conservación del contenido (de las tuplas)

Para toda extensión r de R , la combinación (*join*) de las relaciones resultantes r_i ha de producir la relación origen r , es decir:

$$\begin{aligned} & \bigstar_{i=1}^n r_i = r \end{aligned}$$

donde r_i representa el conjunto de extensiones de los esquemas R_i correspondientes a la extensión r de R .

La combinación de las relaciones resultantes se ha de hacer sobre los atributos comunes; en el caso de que no existiesen, la combinación se convertiría en un producto cartesiano.

Es preciso observar que la condición 2 (conservación de contenido) incluye la 1 (conservación de los atributos), diciéndose que la descomposición de la relación R en el conjunto de relaciones R_i se ha producido “sin pérdida de información” (SPI⁵). Sin embargo, si el proceso de normalización no se lleva a cabo adecuadamente, se puede verificar la primera condición sin cumplirse la segunda, apareciendo en la combinación de las relaciones resultantes nuevas tuplas que no estaban en la relación origen, **tuplas espurias**, que falsean el contenido de la base de datos, tal como se pone de manifiesto en el siguiente ejemplo.

ESTUDIANTE_NUEVA						BECA_NUEVA			
Cód_Est	Nombre_E	Apellidos	DNI	Dirección	Fecha	Cód_Beca	Nombre	Requisito	Fecha
012323	Roberto	Hens	456367	Antonio López 43	10/10/98	A22321	METRICA	Ing. Téc.	10/10/98
763476	Luis	García	345347	Av. Ciudades 29	12/11/98	B56784	ERASMUS	Ing. Téc.	12/11/98
763476	Luis	García	345347	Av. Ciudades 29	14/10/98	A22321	METRICA	Ing. Téc.	14/10/98
763476	Luis	García	345347	Av. Ciudades 29	15/09/98	G65434	HIMMPA	Ingeniería	15/09/98
012323	Roberto	Hens	456367	Antonio López 43	17/09/98	G65434	HIMMPA	Ingeniería	17/09/98
987765	Gregorio	Celada	885764	Pl. Países 67	21/09/98	B56784	ERASMUS	Ing. Téc.	11/11/98
012323	Roberto	Hens	456367	Antonio López 43	11/11/98	B56784	ERASMUS	Ing. Téc.	10/10/98
987765	Gregorio	Celada	885764	Pl. Países 67	10/10/98	A22321	METRICA	Ing. Téc.	12/11/98
012323	Roberto	Hens	456367	Antonio López 43	12/11/98	A22321	METRICA	Ing. Téc.	17/09/98
232457	Mercedes	García	809234	Río Miño 2	17/09/98				

ESTUDIANTE_NUEVA * BECA_NUEVA									
Cód_Est	Nombre_E	Apellidos	DNI	Dirección	Cód_Beca	Nombre	Requisito	Fecha	
012323	Roberto	Hens	456367	Antonio López 43	A22321	METRICA	Ing. Téc.	10/10/98	TUPLAS ESPURIAS
987765	Gregorio	Celada	885764	Pl. Países 67	A22321	METRICA	Ing. Téc.	10/10/98	
763476	Luis	García	345347	Av. Ciudades 29	B56784	ERASMUS	Ing. Téc.	12/11/98	
012323	Roberto	Hens	456367	Antonio López 43	B56784	ERASMUS	Ing. Téc.	12/11/98	
763476	Luis	García	345347	Av. Ciudades 29	A22321	METRICA	Ing. Téc.	14/10/98	
763476	Luis	García	345347	Av. Ciudades 29	G65434	HIMMPA	Ingeniería	15/09/98	
012323	Roberto	Hens	456367	Antonio López 43	G65434	HIMMPA	Ingeniería	17/09/98	
232457	Mercedes	García	809234	Río Miño 2	G65434	HIMMPA	Ingeniería	17/10/98	
987765	Gregorio	Celada	885764	Pl. Países 67	G65434	HIMMPA	Ingeniería	21/09/98	
012323	Roberto	Hens	456367	Antonio López 43	B56784	ERASMUS	Ing. Téc.	11/11/97	
987765	Gregorio	Celada	885764	Pl. Países 67	B56784	ERASMUS	Ing. Téc.	10/10/97	
012323	Roberto	Hens	456367	Antonio López 43	B56784	ERASMUS	Ing. Téc.	10/10/97	
012323	Roberto	Hens	456367	Antonio López 43	A22321	METRICA	Ing. Téc.	12/11/98	
763476	Luis	García	345347	Av. Ciudades 29	A22321	METRICA	Ing. Téc.	12/11/98	
232457	Mercedes	García	809234	Río Miño 2	A22321	METRICA	Ing. Téc.	17/09/98	
012323	Roberto	Hens	456367	Antonio López 43	A22321	METRICA	Ing. Téc.	17/09/98	

Figura 5.2. Ejemplo de descomposición con pérdida de información (aparición de tuplas espurias)

Supongamos la relación origen ESTUDIANTE_SOLICITA_BECA (ver figura 5.1) y las relaciones resultantes de su descomposición (por proyección dejando como

⁵ Cuando una transformación se realiza sin pérdida de información se utiliza muchas veces la expresión inglesa *lossless join*, abreviadamente LLJ.

atributo común la fecha) ESTUDIANTE_NUEVA y BECA_NUEVA que se muestran en la figura 5.2. Observamos que no se conserva el contenido en la descomposición de la relación origen, lo que provoca, al combinar las relaciones resultantes, la aparición de tuplas espurias, señaladas en negrita en la figura.

Si se hubiera hecho una descomposición correcta, tal como aparece en la figura 5.3, la combinación de las tres relaciones sí daría como resultado la relación original.

La transformación sin pérdida de información se basa en la extensión de una relación, por lo que se podría cumplir para ciertas extensiones de la base de datos y no para otras, lo que exigiría algo inviable como es su comprobación ante cada actualización de la base de datos. Es, por tanto, fundamental buscar una condición invariante en el tiempo, es decir que se apoye en el esquema en lugar de en su extensión, que si se cumple implique que una transformación se ha realizado sin pérdida de información. Esta condición está relacionada con las dependencias funcionales, y existe una estrecha conexión entre dependencias funcionales y transformación de esquemas sin pérdida de información.

RISSANEN (1976) propone una descomposición sin pérdida de información, apoyándose en las dependencias funcionales y en las claves (denominada descomposición en **proyecciones independientes**), que presentaremos en el último epígrafe en el que se estudia la normalización por el método de descomposición.

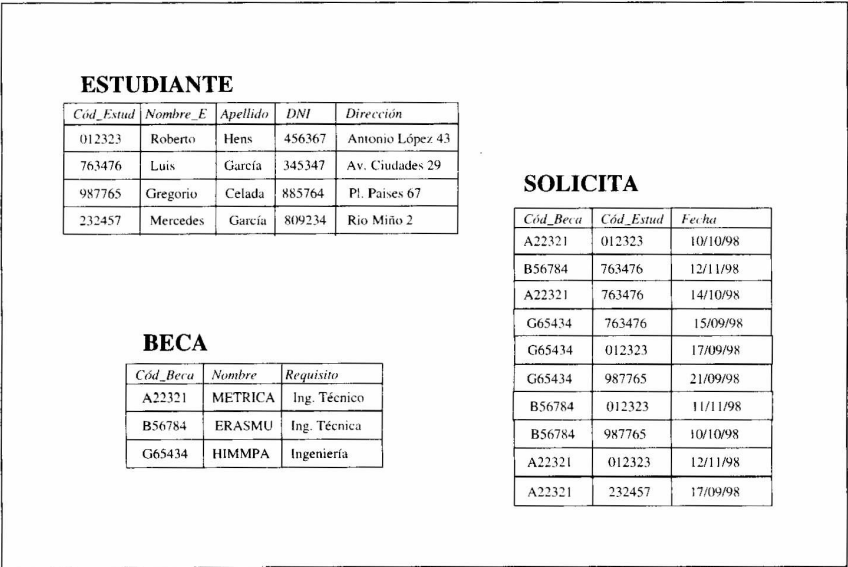


Figura 5.3. Ejemplo de descomposición correcta (sin pérdida de información)

También otros autores, como ULLMAN (1988), DELOBEL (1982), YANG (1985), etc., proponen algoritmos para comprobar que una transformación se ha llevado a cabo sin pérdida de información⁶.

2.2. Conservación de las dependencias

Al ser las dependencias restricciones que recogen la semántica de nuestro mundo real deben estar reflejadas en el esquema de nuestra base de datos. En consecuencia, la transformación del esquema origen R en un conjunto de esquemas R_i debe llevarse a cabo sin pérdida de estas dependencias o, lo que es lo mismo, el conjunto de dependencias funcionales de partida debe ser equivalente al conjunto de dependencias funcionales de los esquemas resultantes (**equivalencia de dependencias**).

Apoyándonos en el concepto de equivalencia de dependencias definido anteriormente, podemos entonces decir que, en el proceso de transformación del esquema origen $R (A, DF)$ en un conjunto de esquemas $R_i (A_i, DF_i)$, se han conservado las dependencias⁷ si se cumple:

$$\left(\bigcup_{i=1}^n DF_i \right)^+ = DF^+$$

Nos encontramos ante un problema computacional, porque, como ya hemos advertido, el cálculo del cierre de un conjunto de dependencias consume mucho tiempo, aun para un número pequeño de dependencias (es exponencial con el número de dependencias de partida), pero ya hemos expuesto un proceso algorítmico, que podemos aplicar aquí, para calcular la equivalencia de dos conjuntos de dependencias funcionales que reduce la complejidad del problema, consiguiéndose una mayor eficiencia.

3. DEFINICIÓN FORMAL DE LAS TRES PRIMERAS FORMAS NORMALES

La tercera propiedad que debe cumplir el conjunto R_i de esquemas resultantes en un proceso de descomposición es que estas relaciones alcancen un nivel de normalización superior al del esquema origen R , a fin de eliminar en lo posible las redundancias y, por tanto, las anomalías de actualización. Como decíamos al principio de este epígrafe, los esquemas R_i , además de ser **equivalentes** a R , deben ser **mejores**

⁶ En el capítulo 7 y en el disquete que acompaña a este libro aparecen los algoritmos que efectúan la comprobación.

⁷ Nos estamos refiriendo a las dependencias funcionales puesto que son las que estamos tratando en este capítulo, pero lo mismo ocurre cuando se considera el conjunto de todas las dependencias (funcionales, multivaluadas y de combinación).

en el sentido de haber alcanzado una forma normal superior al esquema de partida, con menos anomalías de actualización.

Se dice que un esquema de relación está en una determinada forma normal si satisface un cierto conjunto específico de restricciones.

Cuanto más alta sea la forma normal en la que se encuentran los esquemas de relación, menores serán los problemas que aparecen en el mantenimiento de la base de datos.

Codd propuso inicialmente tres formas normales basadas en las dependencias funcionales: primera (1FN), segunda (2FN) y tercera forma normal (3FN), CODD (1970).

La idea que perseguía Codd con su propuesta era la de mostrar las ventajas de las relaciones en 3FN respecto a las relaciones en forma normal inferior: mínima redundancia y mínimas anomalías al actualizar la base de datos.

Debido a que aún persistían los problemas en relaciones en 3FN, Codd, en 1974, introdujo una definición más restrictiva de la tercera forma normal, que se denominó Forma Normal de Boyce-Codd (FNBC).

Fagin introduce la cuarta forma normal (4FN) —FAGIN (1977)— y posteriormente la quinta (5FN) —FAGIN (1979)— basadas en otro tipo de dependencias distintas de las funcionales: las dependencias multivaluadas y las dependencias de proyección-combinación, respectivamente.

Cuando un esquema de relación está en una Forma Normal, implícitamente también está en las formas normales inferiores a ésta. Es decir, un esquema de relación en FNBC, está en 3FN, 2FN y 1FN. Lo contrario no es cierto, un esquema de relación en 2FN puede no estar en 3FN.

Vamos ahora a formalizar la definición de las tres primeras formas normales y de la de Boyce y Codd, que son las que se apoyan en las dependencias funcionales.

3.1. Primera forma normal (1FN)

La primera forma normal (1FN) es una restricción inherente al modelo relacional, por lo que su cumplimiento es obligatorio y afecta al número de valores que pueden tomar los atributos de una relación.

Recordemos que para que una tabla pueda ser considerada una relación no debe admitir grupos repetitivos, esto es, debe estar en *primera forma normal*. En el ejemplo

de la figura 5.4 se puede observar que si un estudiante solicita más de una beca se tienen grupos repetitivos, y para pasar a 1FN habrá que repetir el resto de atributos de la tupla para cada uno de los valores del grupo repetitivo.

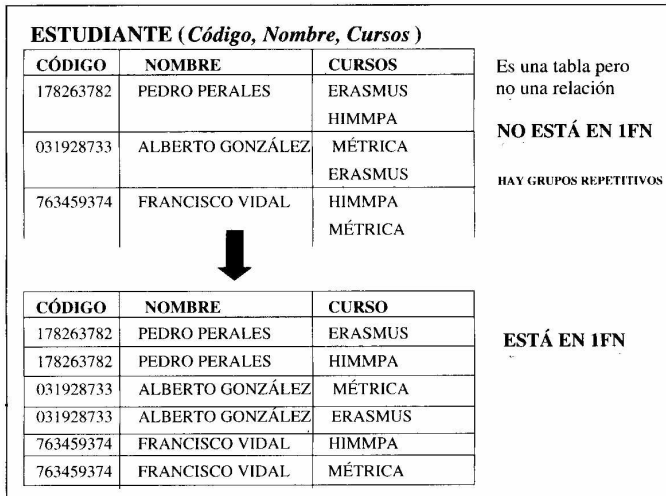


Figura 5.4. 1FN y grupos repetitivos

Definición:

Se dice que una relación está en 1FN cuando cada atributo sólo toma un valor del dominio simple subyacente.

3.2. Segunda forma normal (2FN)

La segunda forma normal (2FN) está basada en el concepto de dependencia plena y en las interrelaciones existentes entre los atributos principales (que se encuentran en alguna de las claves) y no principales (que no se encuentran en ninguna clave) de una relación.

Definición:

Se dice que una relación está en 2FN si:

- Está en 1FN.
- Cada atributo no principal tiene dependencia funcional completa respecto de cada una de las claves.

La segunda forma normal no se cumple cuando algún atributo no principal depende funcionalmente de algún subconjunto de la clave.

Se puede afirmar que cualquier relación binaria se encuentra siempre en 2FN; así como también está en 2FN cualquier relación en la que todas las claves son simples, es decir, contienen un solo atributo. Asimismo, está en 2FN cualquier relación en la que todos sus atributos son **principales**, es decir, forman parte de alguna clave.

Siempre es posible transformar un esquema de relación que no esté en 2FN, en esquemas de relación en 2FN, sin que se produzca pérdida de información ni de dependencias.

Sea el esquema de relación ESTUDIANTE_BECA (AT, DEP) donde:

$$AT = \{Cód_Estudiante, Cód_Beca, Fecha_Sol, Título\}$$

$$DEP = \{Cód_Estudiante, Cód_Beca \rightarrow Fecha_Sol \\ Cód_Estudiante \rightarrow Título\}$$

que refleja las becas que solicitan los estudiantes, la fecha en que lo han hecho y la titulación del estudiante.

La clave de la relación ESTUDIANTE_BECA es (*Cód_Estudiante*, *Cod_Beca*), se puede observar que el atributo *Título* no es un hecho (una información) acerca de la totalidad de la clave, sino acerca de parte de ella (en este caso del atributo *Cód_Estudiante*). Esta relación no está en 2FN.

Transformamos la relación ESTUDIANTE_BECA en las relaciones ESTUDIANTE_BECA1 y ESTUDIANTE que ya sí se encuentran en 2FN:

ESTUDIANTE_BECA1 (AT1, DEP1) donde:

$$AT1 = \{Cód_Estudiante, Cód_Beca, Fecha_Sol\}$$

$$DEP1 = \{Cód_Estudiante, Cód_Beca \rightarrow Fecha_Sol\}$$

y ESTUDIANTE (AT2, DEP2) donde:

$$AT2 = \{Cód_Estudiante, Título\}$$

$$DEP2 = \{Cód_Estudiante \rightarrow Título\}$$

3.3. Tercera forma normal (3FN)

La tercera forma normal (3FN) está basada en el concepto de dependencia transitiva.

Definición:

Un esquema de relación R está en tercera forma normal si, y sólo si:

- Está en 2FN.
- No existe ningún atributo no principal que dependa transitivamente de alguna clave de R.

La tercera forma normal no se cumple cuando existen atributos no principales que dependen funcionalmente de otros atributos no principales.

Se puede afirmar que toda relación binaria se encuentra automáticamente en 3FN, así como toda relación cuyos atributos son todos principales, o bien cuando hay un único atributo no principal.

Siempre es posible transformar un esquema de relación que no está en 3FN, en esquemas de relación en 3FN, sin que se produzca pérdida de información ni de dependencias funcionales.

Sea el esquema de relación ESTUDIANTE (AT, DEP) donde:

$AT = \{Cód_Estudiante, Cód_Proyecto, Nombre_Proyecto\}$

$DEP = \{Cód_Proyecto \rightarrow Nombre_Proyecto$
 $Cód_Estudiante \rightarrow Cód_Proyecto\}$

La única clave del esquema de relación es el atributo *Cód_Estudiente*. El atributo *Nombre_Proyecto* es un hecho acerca del atributo *Cód_Proyecto*, atributo que no forma parte de la clave. Por lo tanto, este esquema de relación no está en 3FN (está en 2FN).

Se puede transformar la relación ESTUDIANTE en las relaciones ESTUDIANTE1 y PROYECTO que sí se encuentran ya en 3FN.

ESTUDIANTE1 (AT1,DEP1) donde:

$AT1 = \{Cód_Estudiante, Cód_Proyecto\}$

$DEP1 = \{Cód_Estudiante \rightarrow Cód_Proyecto\}$

PROYECTO (AT2,DEP2) donde:

AT2={ *Cód_Proyecto*, *Nombre_Proyecto* }

DEP2={ *Cód_Proyecto* $\rightarrow$ *Nombre_Proyecto* }

3.4. Forma normal de Boyce-Codd (FNBC)

Las tres formas normales que acabamos de exponer fueron las propuestas originariamente por CODD (1970), pero, como ya hemos señalado, con el paso del tiempo se mostraron insuficientes para afrontar ciertos problemas en relaciones que presentaban varias claves candidatas compuestas que se solapaban. Por ello, en 1974, Boyce y Codd definieron la llamada forma normal que lleva su nombre (FNBC), ya descrita por Heath, aunque con ligeras diferencias, en 1971. Se trata de una redefinición más estricta de la 3FN.

Se dice que una relación se encuentra en FNBC si, y sólo si, todo determinante es una clave candidata.

Para seguir con nuestro ejemplo de la universidad, si consideráramos que los nombres de los cursos no pueden repetirse, y que un estudiante obtiene una calificación en cada curso al que asiste, la relación

ASISTE (*Cód_Curso*, *Nom_Curso*, *Cód_Estudiente*, *Calificación*)

tendría las siguientes dependencias funcionales:

Cód_Curso $\leftrightarrow$ *Nom_Curso*,

Cód_Curso, *Cód_Estudiente* $\rightarrow$ *Calificación*

y por tanto, tendría dos claves candidatas: {*Cód_Curso*, *Cód_Estudiente*} y (*Nom_Curso*, *Cód_Estudiente*). Esta relación está en 3FN (todos sus atributos, menos uno, son principales), sin embargo tiene anomalías de actualización, ya que se repetiría el nombre y el código de los cursos por cada estudiante que asistiese a ellos; el problema es debido a que la relación ASISTE no se encuentra en FNBC, ya que tanto *Cód_Curso* como *Nom_Curso* son determinantes, pero no son claves candidatas de la relación.

Si una relación cuyas claves no están solapadas se encuentra en 3FN está también en FNBC. Ahora bien, la existencia de claves candidatas solapadas no lleva siempre consigo que la relación no esté en FNBC. Consideremos el esquema:

SE_MATRICULA1 (*Cód_Curso*, *Cód_Edición*, *Cód_Estudiente*, *Fecha*)

donde

Cód_Curso, Cód_Edición, Cód_Estudiente → *Fecha*
Cód_Edición, Cód_Estudiente, Fecha → *Cód_Curso*

las claves candidatas de esta relación son (*Cód_Curso, Cód_Edición, Cód_Estudiente*) y (*Cód_Edición, Cód_Estudiente, Fecha*), que se solapan ya que comparten los atributos *Cód_Edición* y *Cód_Estudiente*; sin embargo, debido a que los únicos determinantes son los dos descriptores anteriores, que son claves candidatas, la relación sí se encuentra en FNBC.

Se puede afirmar que toda relación binaria está en FNBC.

No siempre es posible transformar un esquema de relación que no está en FNBC en esquemas de relación en FNBC sin que se produzca pérdida de dependencias funcionales. Sí se puede asegurar que la transformación se puede producir siempre sin pérdida de información.

Sea el esquema de relación CLASE (AT,DEP) donde:

AT={ *Cód_Estudiente, Cód_Profesor, Materia* }

DEP={ *Cód_Estudiente, Materia* → *Cód_Profesor*
Cód_Profesor → *Materia* }

Este esquema de relación tiene dos claves candidatas: (*Cód_Estudiente, Materia*) y (*Cód_Estudiente, Cód_Profesor*).

La relación así definida está en 3FN —todos sus atributos son principales— pero no está en FNBC, puesto que el determinante *Cód_Profesor* no es una clave candidata de la relación.

Se puede transformar la relación CLASE en las relaciones CLASE1 y CLASE2 que ya sí se encuentran en FNBC:

CLASE1 (AT1,DEP1) donde:

AT1={ *Cód_Estudiente, Cód_Profesor* }
 DEP1={ }

CLASE2 (AT2,DEP2) donde:

AT2={ *Cód_Profesor, Materia* }
 DEP2={ *Cód_Profesor* → *Materia* }

La dependencia *Cod_Estudiente, Materia* → *Cód_Profesor* se ha perdido en la transformación anterior, ya que no es posible deducirla del conjunto de dependencias de los esquemas resultantes. A pesar de ello, ésta es la mejor descomposición de las tres posibles, ya que en las otras dos se pierde además información.

Podemos, por tanto, deducir que no siempre es posible llegar a FNBC sin pérdida de dependencias funcionales, por eso existen autores que, como KORTH y SILBERSCHATZ (1986), no aconsejan pasar a la FNBC, sino detenerse en la 3FN; otros, como DATE (1990), prefieren, sin embargo, seguir con el proceso de normalización hasta sus formas más avanzadas. En el próximo epígrafe, cuando analicemos el método de descomposición, estudiaremos más detenidamente la posible pérdida de dependencias al descomponer una relación hasta FNBC.

Como señala SALTOR (1980), en el paso de la 3FN a la FNBC “puede ser posible encontrar soluciones más satisfactorias que las hasta ahora propuestas mediante la explicitación de atributos o relaciones ocultos y semánticamente significativos”. Por tanto, puede compensar profundizar en el estudio de la semántica de las relaciones con la posibilidad de crear atributos o relaciones que nos lleven a un esquema relacional más adecuado.

Puede ocurrir que ciertas relaciones que se encuentran en FNBC presenten todavía redundancias y anomalías, pero éstas ya no están basadas en las DF; para evitarlas se han definido la cuarta y la quinta forma normal que se estudiarán en el próximo capítulo y que están relacionadas con las dependencias multivaluadas y las de combinación.

4. DOS ENFOQUES DE DISEÑO RELACIONAL: ANÁLISIS Y SÍNTESIS

Existen dos escuelas que implican dos grupos de algoritmos a la hora de aplicar la teoría de la normalización: la que propugna los métodos llamados de **análisis** o **descomposición**, y la que aboga por el procedimiento de **síntesis**.

El método de descomposición fue propuesto primeramente por Codd, que, ya desde su primer trabajo, va descomponiendo una relación paso a paso hasta alcanzar la 3FN; posteriormente, RISSANEN (1973) desarrolla teóricamente el método, que es generalizado por FAGIN (1977) y por ZANIOLO (1981) para incluir dependencias multivaluadas y de combinación. En el análisis se parte del esquema de la relación universal (que contiene todos los atributos del universo del discurso y las dependencias existentes entre ellos)⁸ y se va **descomponiendo**, por sucesivas proyecciones que han de cumplir los principios de conservación de la información y de las

⁸ También se puede aplicar a cada una de las relaciones que constituyen un esquema relacional, las cuales se han podido obtener a partir de un esquema E/R.

dependencias; los esquemas resultantes serán cada vez de menor grado y estarán en formas normales cada vez más avanzadas, es decir, se irán reduciendo las anomalías.

El proceso se suele llevar a cabo mediante lo que se llama un árbol de descomposición, el cual se presenta en la figura 5.5, donde $A_1, A_2, \dots, A_n$ representan el conjunto de atributos y las $f_1, f_2, \dots, f_m$ el conjunto de dependencias⁹. El proceso de descomposición termina cuando las relaciones se encuentran en la forma normal deseada, o cuando la continuación del proceso supondría una pérdida de dependencias, si es objetivo prioritario la conservación de las mismas.

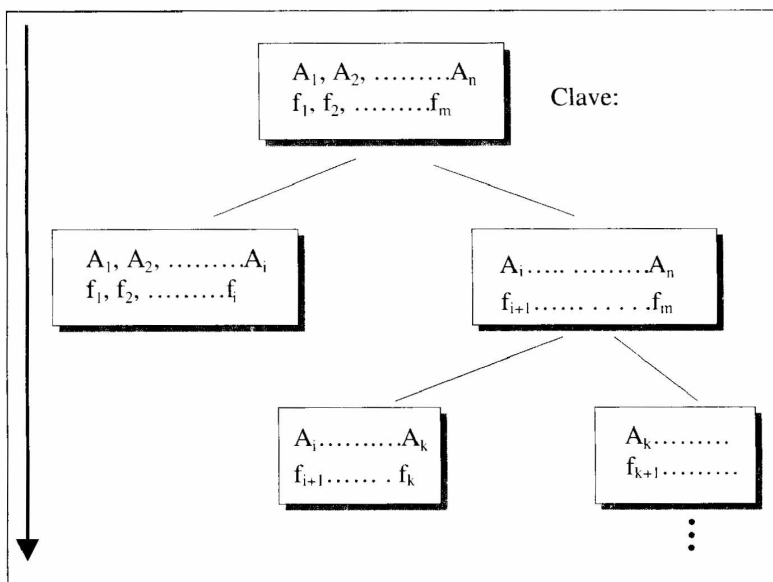


Figura 5.5. Árbol de descomposición

En 1976, Bernstein propone un método de diseño relacional alternativo al análisis que se conoce con el nombre de *síntesis*.

Así como las técnicas de análisis van **descomponiendo** un esquema de relación en otros, cada vez de menor grado, la síntesis recorre el camino inverso, obteniendo (**sintetizado**) relaciones a partir de un conjunto de atributos y de dependencias funcionales; es decir, la síntesis busca agrupar atributos a fin de tener en una relación toda la información correspondiente a un objeto (entidad o interrelación) del mundo real, mientras que en el análisis se pretende separar la información referente a objetos distintos.

⁹ Si bien en este capítulo sólo se han estudiado las dependencias funcionales, el método es también aplicable a las dependencias multivaluadas y de combinación.

Se puede representar gráficamente los métodos de síntesis tal como aparece en la figura 5.6; a veces, el proceso de síntesis se apoya en la teoría de grafos —véase, por ejemplo, YANG (1986) —.

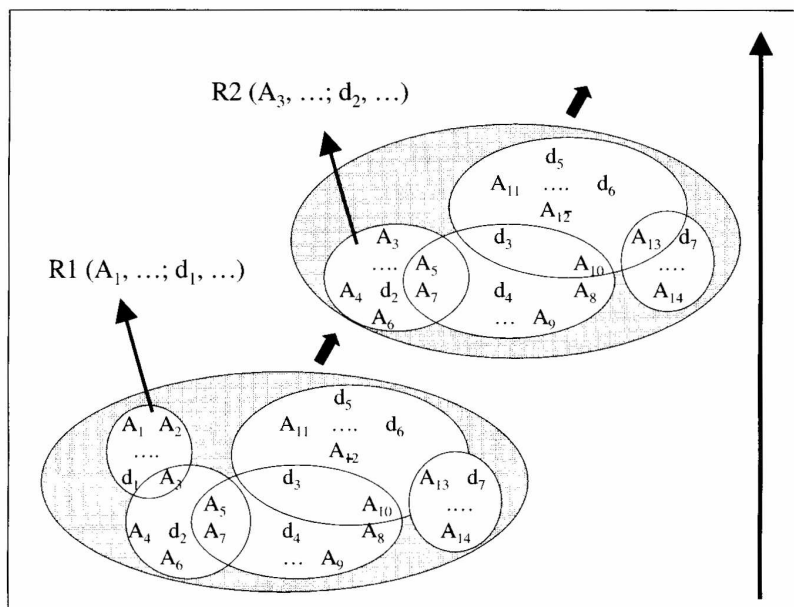


Figura 5.6. Representación del proceso de síntesis

Análisis y síntesis tienen, por tanto, la misma finalidad, y ambos procesos se apoyan en el mismo concepto de dependencias y de **recubrimiento irredundante**, pero siguen caminos opuestos para llegar a obtener esquemas que respondan a una determinada forma normal; además, el primero toma también en consideración las dependencias multivaluadas y de proyección/combinación, por lo que permite obtener relaciones en 5FN, mientras que la síntesis, tal como se propone en el algoritmo de Bernstein, sólo nos asegura llegar hasta la 3FN; aunque muchas veces, si no hay claves candidatas solapadas y sólo existen dependencias funcionales, esto signifique que el esquema relacional resultante está también en FNBC, en 4FN y en 5FN, pero el proceso de síntesis de Bernstein no nos puede decir nada al respecto.

En la figura 5.7 se presentan de forma resumida las características principales de ambos métodos. El método de síntesis parte de las dependencias funcionales para llegar hasta la 3FN, conservando la información y las dependencias; y, teóricamente, hasta FNBC, en cuyo caso no puede asegurarse que se conserve la información. Por lo que respecta al método de análisis o descomposición, se parte de las dependencias funcionales, si sólo se desea llegar hasta la 3FN o FNBC, o bien de las dependencias

funcionales, multivaluadas y de combinación para llegar hasta la 5FN; si se llega a la FNBC no se puede asegurar que no haya habido pérdida de dependencias.

	SÍNTESIS		ANÁLISIS		
Dependencias departida	DF		DF		DF, DM, DC
Forma Normal alcanzada	3FN	FNBC	3FN	FNBC	5FN
Conservación informac.	SÍ	no se asegura	SÍ	SI	SÍ
Conserv Dependec Func	SÍ	SÍ	SÍ	no se asegura	no se asegura

Figura 5.7. Comparación de los métodos de Síntesis y Análisis

4.1. Análisis

El método de análisis permite analizar una estructura relacional existente (podría ser el esquema de la relación universal que contiene todos los atributos, o bien un conjunto de esquemas de relación), determinando su nivel de normalización y descomponiéndola en nuevas estructuras relacionales más regulares que cumplan ciertas propiedades.

Se trata, por tanto, de pasar unos determinados *test* a un esquema de relación para comprobar si se encuentra o no en una determinada forma normal y, en caso negativo, ir transformando en pasos sucesivos dicho esquema en otros de menor grado, utilizando para ello el operador de proyección. Se pasa así a obtener esquemas en formas normales más avanzadas que el esquema de partida.

El problema de determinar si una relación está en 3FN exige encontrar todas las claves de la misma, a fin de obtener los atributos principales y los que no lo son; como se demuestra en BEERI (1979) se trata de un problema NP-completo, de ahí la dificultad del proceso. La determinación de si una relación está en FNBC sólo exige comprobar si todos los determinantes son clave de la relación, siendo polinomial la complejidad de los correspondientes algoritmos; en PARKER (1980) se expone un test que comprueba si una relación está en FNBC.

A veces, cuando se habla de la teoría de la normalización, algunos autores, en nuestra opinión de forma poco precisa, se refieren exclusivamente al análisis, es decir, al diseño por descomposición.

La normalización, según este método, consiste, por tanto, en descomponer mediante proyecciones las relaciones que presentan anomalías y redundancias en otras de menor grado, intentando conservar siempre la información y las dependencias funcionales.

Formalmente, la descomposición de un esquema de relación $R(A, DF)$ consiste en la sustitución de dicho esquema por un conjunto de proyecciones del mismo:

$$R_1, R_2, \dots, R_p \text{ donde } R_i(A_i, DF_i)$$

tales que, como dijimos en el epígrafe anterior, el conjunto resultante sea **equivalente y mejor** que el esquema origen.

Para que la descomposición se lleve a cabo sin pérdida de información se ha de cumplir que:

$$R = R_1 * R_2 * \dots * R_p$$

para toda extensión de R (descomposición SPI).

Asimismo, la descomposición debe hacerse sin pérdida de dependencias funcionales:

$$(U DF_i)^+ = DF^+$$

4.1.1. DESCOMPOSICIÓN EN PROYECCIONES INDEPENDIENTES

RISSANEN (1976) ofrece unos principios que nos permiten saber si una determinada descomposición es correcta, es decir, si conserva la información y las dependencias funcionales; para ello introduce el concepto de *proyecciones independientes*.

Sea R una relación y R_1 y R_2 dos de sus proyecciones, se dice que dichas proyecciones son independientes si, y sólo si,

- 1) Sus atributos comunes son la clave primaria de, al menos, una relación.
- 2) Cada dependencia funcional en R puede deducirse de las de R_1 y R_2 .

Así, por ejemplo, dada la relación

$$\begin{aligned} \text{ESTUDIANTE } (\{ \text{Cód_Estudiante}, \text{Cód_Programa} \}, \\ \{ \text{Cód_Estudiante} \rightarrow \text{Cód_Programa} \\ \text{Cód_Programa} \rightarrow \text{Departamento} \}) \end{aligned}$$

que no está en 3FN, existen tres posibilidades de descomposición:

1)

ESTUDIANTE1 ($\{Cód_Estudiante, Departamento\}$,
 $\{Cód_Estudiante \rightarrow Departamento\}$)

ESTUDIANTE2 ($\{Cód_Programa, Departamento\}$,
 $\{Cód_Programa \rightarrow Departamento\}$)

que atenta contra el primer principio de Rissanen ya que el atributo común, *Departamento*, no es clave primaria de ninguna de las dos relaciones, puesto que en la primera la clave es *Cód_Estudiante*, y en la segunda, *Cód_Programa*.

En esta descomposición se ha perdido información, ya que al combinar las relaciones ESTUDIANTE1 y ESTUDIANTE2 no se obtiene la relación origen, sino que aparecen **tuplas espurias**, que surgen de la combinación de cada estudiante con todos los programas de doctorado existentes en ese Departamento, lo que es falso, como ya hemos visto anteriormente, ya que los estudiantes no siguen todos los programas de doctorado existentes en el Departamento sino sólo uno de ellos.

2)

ESTUDIANTE3 ($\{Cód_Estudiante, Cód_Programa\}$,
 $\{Cód_Estudiante \rightarrow Cód_Programa\}$)

ESTUDIANTE4 ($\{Cód_Estudiante, Departamento\}$,
 $\{Cód_Estudiante \rightarrow Departamento\}$)

que cumple el primer principio de Rissanen, puesto que el atributo común, *Cód_Estudiante*, es clave en las dos relaciones, aunque atenta contra el segundo principio al perderse la dependencia funcional existente entre $Cód_Programa \rightarrow Departamento$, que no puede deducirse de $\{Cód_estudiante \rightarrow Cód_Programa, Cód_Estudiante \rightarrow Departamento\}$.

Esta pérdida de dependencias puede originar problemas en la base de datos, ya que en el esquema no se obliga a que cada *Cód_Programa* dependa de un único Departamento, para lo que se deberían establecer controles adicionales por programa.

3)

ESTUDIANTE5 ($\{Cód_Estudiante, Cód_Programa\}$,
 $\{Cód_Estudiante \rightarrow Cód_Programa\}$)

ESTUDIANTE6 ($\{Cód_Programa, Departamento\}$,
 $\{Cód_Programa \rightarrow Departamento\}$)

que cumple los dos principios de Rissanen, ya que

- a) El atributo común, *Cód_Programa*, es clave de una relación, ESTUDIANTE6
- b) Cada dependencia funcional de la relación ESTUDIANTE se encuentra o se puede deducir de las presentes en ESTUDIANTE5 y ESTUDIANTE6; así de $\{Cód_Estudiante \rightarrow Cód_Programa, Cód_Programa \rightarrow Departamento\}$ por el axioma de transitividad, $Cód_Estudiante \rightarrow Departamento$, que son las tres dependencias funcionales presentes en la relación ESTUDIANTE.

Por tanto, de las tres descomposiciones posibles únicamente la tercera cumple los dos principios de Rissanen, siendo una descomposición sin pérdida de información ni de dependencias y, por ello, la mejor.

4.1.2. DESCOMPOSICIÓN HASTA FNBC

Es preciso que la descomposición de relaciones se efectúe en proyecciones independientes. Para las tres primeras formas normales esto siempre es posible, pero no siempre lo es si se desea llegar a la forma normal del Boyce y Codd, ya que para ello, a veces, es preciso perder dependencias o información. Veamos dos ejemplos de descomposición en FNBC, en el primero de los cuales la descomposición puede llevarse a cabo en proyecciones independientes, es decir, sin pérdida de información ni de dependencias, lo que no ocurre en el segundo.

Ejemplo 1. Sea la relación

DOCTORES ($\{Cód_Profesor, Nombre_Profesor, Cód_Estudiante, Nota\}$
 $\{Cód_Profesor, \leftrightarrow Nombre_Profesor$
 $Cód_Profesor, Cód_Estudiante \rightarrow Nota\}$)

que está en 2FN ya que *Nota*, que es el único atributo no principal, depende plenamente de las dos claves de la relación (*Cód_Profesor*, *Cód_Estudiante*) y (*Nombre_Profesor*, *Cód_Estudiante*). También está en 3FN al existir solamente un atributo no principal. Sin embargo, no está en FNBC, ya que tanto *Cód_Profesor* como *Nombre_Profesor* son determinantes, pero no claves candidatas.

La descomposición de esta relación se podría hacer de la siguiente manera.

DOCTOR1 ($\{Cód_Profesor, Nombre_Profesor\}$
 $\{Cód_Profesor \leftrightarrow Nombre_Profesor\}$)

NOTA ($\{Cód_Profesor, Cód_Estudiante, Nota\}$
 $\{Cód_Profesor, Cód_Estudiante \rightarrow Nota\}$)

en la que se conservan tanto las dependencias como la información.

Ejemplo 2. Sea la relación:

CURSO (*{Cód_Curso, Cód_Profesor, Texto}*; DF)

donde suponemos que un determinado curso con un texto sólo lo imparte un profesor y que un profesor utiliza un solo texto independientemente del curso que imparte; las DF serían:

$$\{Cód_Curso, Texto \rightarrow Cód_Profesor, Cód_Profesor \rightarrow Texto\}$$

Esta relación se encuentra en 2FN y 3FN, ya que todos los atributos son principales. Las claves candidatas de la relación son (*Cód_Curso, Texto*) y (*Cód_Curso, Cód_Profesor*), mientras que los determinantes son (*Cód_Curso, Texto*) y (*Cód_Profesor*) por lo que no todo determinante es clave candidata de la relación, no encontrándose ésta, por tanto, en FNBC.

La relación puede descomponerse de tres formas distintas:

1) CURSO1 (*{Cód_Curso, Texto}*, { })

CURSO2 (*{Texto, Cód_Profesor}*,
{Cód_Profesor $\rightarrow$ Texto})

En esta descomposición no sólo se pierden dependencias funcionales, sino que también se pierde información, ya que si combinamos las relaciones CURSO1 y CURSO2, aparecen tuplas espurias al combinar cada curso con todos los profesores que utilizan cada texto, lo que es debido a que el atributo común *Texto* no es clave de ninguna de las dos relaciones.

2) CURSO3 (*{Cód_Curso, Cód_Profesor}*, { })

CURSO4 (*{Cód_Curso, Texto}*, { })

En esta descomposición, además de perder dependencias funcionales, también se pierde información, ya que si combinamos las relaciones CURSO3 y CURSO4, aparecen tuplas espurias al combinar a través de *Cód_Curso* profesores con textos que no les corresponden. (Recordemos que hemos supuesto que un mismo curso se puede impartir con textos distintos.)

3) CURSO5 (*{Cód_Curso, Cód_Profesor}*, { })

CURSO6 (*{Cód_Profesor, Texto}*,
{Cód_Profesor $\rightarrow$ Texto})

Esta descomposición, aunque resulta la mejor de las tres, sigue produciendo la pérdida de la dependencia funcional $Cód_Curso, Texto \rightarrow Cód_Profesor$, presente en la relación original.

Como hemos visto en este ejemplo, la descomposición no puede llevarse a cabo sin pérdida de dependencias funcionales.

4.1.3. EL PROCESO DE DESCOMPOSICIÓN¹⁰

Supongamos la relación con esquema:

$R(A, DF)$

donde A es un conjunto de atributos y DF un conjunto de dependencias funcionales¹¹. Supongamos que esta relación sufre anomalías, por lo que se desea avanzar en su nivel de normalización aplicándole un proceso de descomposición. Los pasos a seguir serían:

- 1) Hallar un recubrimiento minimal DF^m .
- 2) Determinar la(s) clave(s), así como los atributos principales y no principales.
- 3) Identificar la FN en que se encuentra la relación.

Si se desea llegar a una forma normal más avanzada:

- 4) Agrupar las DF que tengan el mismo implicante.
- 5) Obtener proyecciones independientes sobre cada una de las dependencias funcionales (o de los grupos), de forma que los atributos que aparecen en la correspondiente dependencia constituyen una nueva relación y el implicante de la dependencia, así como ésta, desaparezcan de la relación origen.
- 6) Proseguir esta descomposición repitiendo el paso 5 hasta que no pueda continuarse porque todas las dependencias estén implicadas por una clave (ya hemos advertido que, a veces, para llegar hasta FNBC hay que perder dependencias; en este caso es decisión del diseñador parar el proceso en la 3FN o avanzar hasta FNBC con el inconveniente señalado¹²).

En el proceso de descomposición es relevante el orden en el que se van tomando las dependencias funcionales, ya que éste debe ser tal que se consiga que los atributos

¹⁰ En el capítulo 7 se incluye un algoritmo de descomposición.

¹¹ El proceso de descomposición en el que se incluyen también dependencias multivaluadas y de combinación es muy parecido al que exponemos a continuación y se presentará en el próximo capítulo.

¹² También es posible profundizar en la semántica de las relaciones, ya que se puede encontrar algún atributo oculto o alguna clave susceptible de descomposición que permitiría una solución más satisfactoria.

que desaparecen de la relación origen sean aquellos que no entren a formar parte de ninguna otra dependencia, ya que en caso contrario se perderían dependencias. Este problema puede observarse en el ejemplo de la figura 5.8, en el que si se comienza la descomposición por la dependencia $a \rightarrow b$, desaparece b de los atributos de R con lo cual se pierde la dependencia funcional $b \rightarrow c$, y la clave que era a en R pasa a ser a, c en R' obteniéndose las relaciones $R1$ y $R2$ y $R3$ que constituyen una descomposición incorrecta con pérdida de una dependencia funcional.

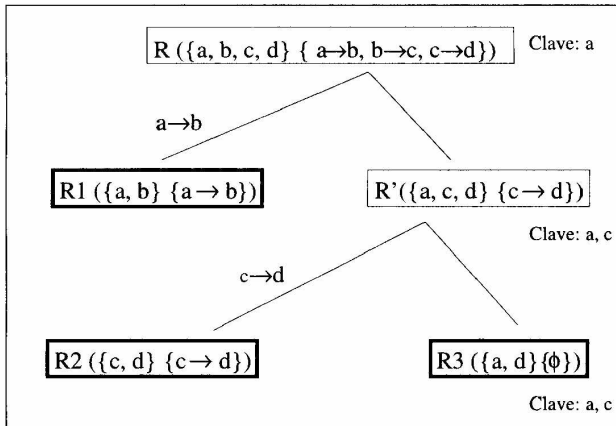


Figura 5.8. Ejemplo de descomposición incorrecta

La descomposición correcta, que puede verse en la figura 5.9, debe comenzar con la dependencia $c \rightarrow d$.

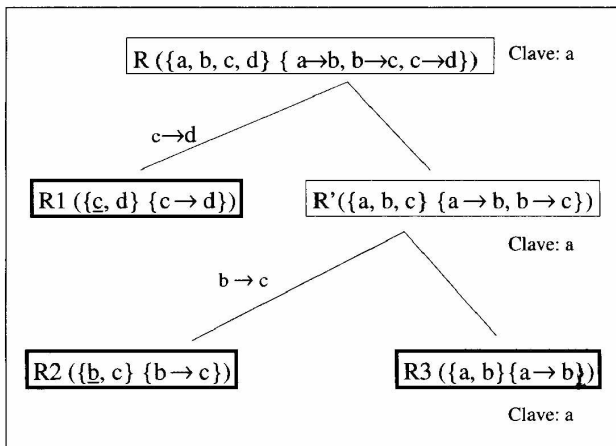


Figura 5.9. Ejemplo de descomposición correcta

Si la descomposición se realiza correctamente, podemos asegurar que siempre será posible llegar a 3FN sin pérdida de información ni de dependencias funcionales, lo que no se puede asegurar si se desea llegar a FNBC.

4.2. Proceso de síntesis

El proceso de síntesis se puede describir, en resumen¹³, de la forma siguiente: Dado el esquema:

$R(A, DF)$

donde A es un conjunto de atributos y DF el conjunto de dependencias *funcionales*¹⁴ que existen entre dichos atributos.

- 1) Se busca un recubrimiento minimal DF^m del conjunto de dependencias funcionales DF .
- 2) Se agrupan las dependencias de DF^m en particiones que tengan el mismo implicante.
- 3) Se forma un esquema de relación R_i para cada partición, el cual tendrá como atributos todos los que aparezcan en la correspondiente partición así como las dependencias funcionales implicadas.
- 4) Si existen atributos que no son implicantes ni implicados en DF^m , se forma un esquema de relación con éstos sin dependencias funcionales.

Alternativamente, se añade la clave de la relación inicial como una relación.

En el siguiente ejemplo se muestra cómo aplicar el proceso de síntesis:

Ejemplo:

Un departamento universitario desea diseñar una base de datos para la gestión de los cursos que imparte durante un cuatrimestre. En la base de datos quiere almacenar los profesores (P), los estudiantes (E), la nota (N) con la que se califica a un alumno en cada asignatura (AS), así como los días de la semana/hora (H) en las que se imparte una asignatura y el aula (AU) (se supone que ni el día/hora ni el aula en los que se imparte una asignatura varían de una semana a otra). Se desea almacenar también el

¹³ El algoritmo completo de síntesis se expone en el capítulo 7.

¹⁴ Se ha de observar que el proceso de síntesis propuesto por Bernstein sólo considera las dependencias funcionales.

teléfono (TL) y el despacho (D) de cada profesor (se supone que no existen teléfonos compartidos por dos profesores y que en cada despacho sólo hay un profesor y un teléfono. Además de los anteriores se dan los siguientes supuestos semánticos:

- a) En un momento dado tanto un estudiante como un profesor sólo pueden estar en un aula.
- b) En un momento dado en un aula sólo se puede impartir una asignatura.
- c) En cada despacho hay un solo teléfono.
- d) Un estudiante no puede asistir a las clases de dos asignaturas en una misma hora.

1. Primero se determinan las dependencias funcionales

- a) $H, E \rightarrow AU; H, P \rightarrow AU$
- b) $H, AU \rightarrow AS$
- c) $D \rightarrow TL$
- d) $H, E \rightarrow AS$
- e) $E, AS \rightarrow N$
- f) $P \rightarrow TL; P \rightarrow D; TI \rightarrow P; D \rightarrow P; D \rightarrow TL$

2. Obtenemos el recubrimiento minimal

$H, E \rightarrow AU; H, P \rightarrow AU; H, AU \rightarrow AS; E, AS \rightarrow N; P \rightarrow D; P \rightarrow TL; TL \rightarrow P;$
 $D \rightarrow P$

(Las dependencias no redundantes entre profesor, despacho y teléfono pueden ser otras)

3. Definimos la forma normal en la que se encuentra la relación

CLAVES: P, D y TL son atributos equivalentes, luego me quedo sólo con P

IMPLICANTES: H, E, P, AU, AS

IMPLICADOS : AU, AS, N

Atributos que no se encuentran en ninguna dependencia funcional: G, T

Existen tres claves: H, E, P, G, T; H, E, D, G, T; H, E, TL, G, T

Atributos no principales: N, AS, AU

N no depende de la totalidad de la clave, luego la relación no está en 2FN

4. Esquema relacional normalizado

Aplicamos el método de síntesis (lo que nos permite asegurar que las relaciones resultantes están, al menos, en 3FN)

R1 (H, E, AU; $H,E \rightarrow AU$)	Aula por estudiante y hora
R2 (H, P, AU; $H,P \rightarrow AU$)	Aula por profesor y hora
R3 (H, AU, AS; $H,AU \rightarrow AS$)	Asignatura por aula y hora
R4 (E, AS, N; $E, AS \rightarrow N$)	Nota por asignatura y estudiante
R5 (P, D, TL; $P \leftrightarrow D; P \leftrightarrow TL$)	Profesor con despacho y teléfono
R6 (H, E, P, G, T)	La clave de la relación

Observaciones: Las relaciones R1, R5 y R6 están, al menos, en 3FN.

FORMAS NORMALES AVANZADAS Y REORGANIZACIÓN DE RELACIONES

En este capítulo se profundiza en la teoría de la normalización en base a dependencias distintas a las funcionales. Se comienza estudiando las dependencias multivaluadas y de combinación a fin de abordar formas normales (4FN y 5FN) más avanzadas que las tratadas en el capítulo anterior. Se presentan, además, las dependencias de dominio/clave y de inclusión; por último se analizan distintas maneras de "reorganizar" las relaciones, como son la desnormalización, el particionamiento horizontal y el particionamiento vertical.

1. SEMÁNTICA DE LOS DATOS Y NUEVOS TIPOS DE DEPENDENCIAS

En el diseño de bases de datos en el modelo relacional, al igual que en otros modelos de datos, lo que se pretende —insistimos en ello— es recoger en el esquema la mayor semántica posible del universo del discurso que se está tratando de modelar, a fin de que el esquema sea una representación fiel de nuestro mundo real.

En el capítulo anterior hemos estudiado un tipo especial de restricciones semánticas sobre los datos, las dependencias funcionales, que nos permiten diseñar esquemas en 2FN, 3FN y FNBC; pero aun las relaciones que se encuentran en FNBC siguen, a veces, presentando redundancias y provocando anomalías en el momento de su actualización. La razón de ello es que existe otro tipo de dependencias entre datos, distintas de las funcionales, que también hay que tener en cuenta en el momento del diseño de una base de datos relacional.

Esto conduce a una generalización de las dependencias funcionales, apareciendo nuevos tipos de dependencias como las multivaluadas, las de combinación, y las correspondientes embebidas. Las dependencias funcionales son un caso especial de las multivaluadas y éstas, a su vez, de las de combinación.

Las dependencias multivaluadas y de combinación, que definiremos en este capítulo, van a dar lugar a nuevas formas normales —la cuarta y la quinta—, continuando así el proceso de normalización que, hasta la FNBC, se vio en el capítulo anterior.

El proceso de concepción de un esquema relacional se va, por tanto, a apoyar en los diferentes tipos de dependencias, ya que si se limita a las dependencias funcionales, sólo se obtendrá, desde un punto de vista teórico, una solución aproximada. Debemos, sin embargo, llamar la atención sobre dos hechos. El primero se refiere a que la solución basada sólo en las DF será tanto más aproximada cuantas menos dependencias multivaluadas y de combinación existan en nuestro universo del discurso; si no existiese ninguna, la FNBC coincidiría con la 5FN y no podríamos tachar la solución de "aproximada". El segundo atañe a **consideraciones prácticas**, ya que el que una solución sólo sea aproximada desde un punto de vista teórico no quiere decir que no pueda ser la más adecuada en la práctica (en los últimos epígrafes de este capítulo volveremos sobre estos temas).

Queremos destacar la vinculación existente entre la semántica de los datos (expresada en el ME/R), la teoría de las dependencias y el diseño relacional, lo que representamos en la figura 6.1, donde puede observarse la correspondencia entre distintos conceptos: la semántica de los datos en el modelo E/R se puede expresar, en general, mediante la teoría de las dependencias (funcionales, multivaluadas, combinación, inclusión), y ésta es el fundamento del diseño relacional. Así, la asociación entre atributos identificadores y no identificadores de una entidad se expresa mediante dependencias funcionales, las cuales son la base para la definición de la segunda y tercera formas normales, así como la de Boyce y Codd. Las interrelaciones de cualquier grado con cardinalidad máxima de 1 en al menos una de las entidades o con atributos propios de la interrelación se expresan también mediante dependencias funcionales, pero en este caso aparecen dependencias de inclusión (integridad referencial). Las interrelaciones de grado superior a 2 con cardinalidad máxima de n en todas las entidades, si pueden descomponerse en otras de menor grado es porque en la relación mediante la cual se expresan aparecen dependencias multivaluadas o de combinación relacionadas con la cuarta y quinta formas normales. Sin embargo, las interrelaciones con cardinalidad n en todas las entidades y sin atributos no pueden expresarse mediante la teoría de las dependencias.

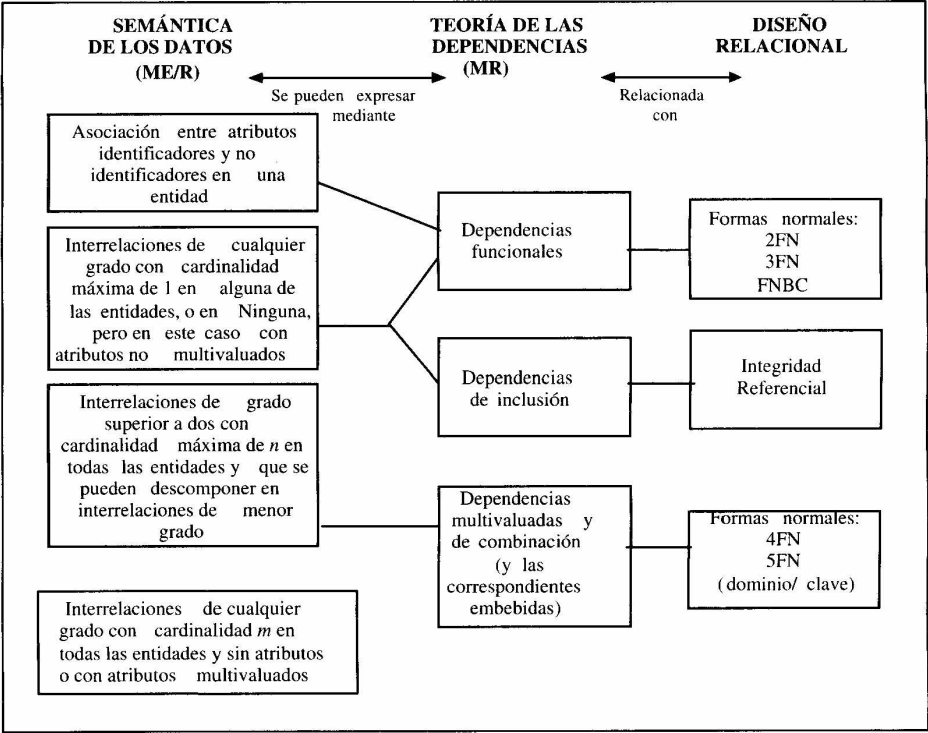


Figura 6.1. Relación entre semántica de los datos (en el ME/R), teoría de las dependencias (en el MR) y diseño relacional

2. DEPENDENCIAS MULTIVALUADAS Y CUARTA FORMA NORMAL

Las dependencias multivaluadas son una generalización de las dependencias funcionales, y así como en estas últimas el valor de un descriptor (implicante) determina un **solo** valor de otro descriptor (implicado), en las multivaluadas aparece un conjunto de valores del implicado en lugar de uno solo; esta situación puede producirse cuando existen grupos repetitivos (atributo multivaluado) y se normaliza la tabla para que esté en 1FN.

Veamos la noción intuitiva de dependencia multivaluada mediante un ejemplo. Supongamos la tabla no normalizada¹ ASIGNATURAS de la figura 6.2. En ella podemos observar que un cierto nombre de asignatura —por ejemplo, "Archivos y

¹ Obsérvese que decimos "tabla", ya que al no estar normalizada, no se trata realmente de una "relación".

BD"— caracteriza un conjunto bien definido de profesores que la imparten —{ Sr. Sánchez, Sra. Hidalgo }— y un conjunto de textos —{ Concepción y Diseño de BD, Fundamentos de BD }—, siendo los profesores independientes de los textos y viceversa, por lo que han de aparecer en la relación que se obtiene al normalizar esta tabla, todas las posibles combinaciones entre los valores de ambos atributos.

ASIGNATURAS		
Nom _Asignatura	Profesor	Texto
Ficheros y BD	{ Sr. Sánchez Sra. Hidalgo }	{ Concepción y Diseño de BD Fundamentos de BD }
BD avanzadas	{ Sra. Hidalgo Sr. Martín }	{ Diseño de BD avanzadas }

Figura 6.2. Tabla no normalizada (con grupos repetitivos)

La normalización de esta tabla daría lugar, por tanto, a la relación que se presenta en la figura 6.3, donde se puede observar una gran cantidad de redundancias, con las consiguientes posibilidades de que aparezcan anomalías de actualización; sin embargo, la relación ASIGNATURAS de la figura 6.3 está en FNBC, ya que no existen en ella dependencias funcionales, y su clave es el conjunto de los tres atributos.

Aunque no haya, en dicha relación, dependencias funcionales, sí aparecen unas determinadas interrelaciones entre los datos; se trata de las **dependencias multivaluadas**, por las que *Nom_Asignatura* se dice "multidetermina" a *Profesor* y también multidetermina a *Texto*.

Las dependencias multivaluadas se producen cuando en una tabla aparecen atributos multivaluados independientes entre sí. Así, en el ejemplo, *Profesor* y *Texto* son atributos multivaluados independientes entre sí, ya que suponemos que en el mundo real que nos ocupa existe una regla que obliga a que todos los profesores que imparten una asignatura utilicen todos los textos correspondientes a dicha asignatura.

ASIGNATURAS		
Nom _Asignatura	Profesor	Texto
Ficheros y BD	Sr. Sánchez	Concepción y Diseño de BD
Ficheros y BD	Sr. Sánchez	Fundamentos de BD
Ficheros y BD	Sra. Hidalgo	Concepción y Diseño de BD
Ficheros y BD	Sra. Hidalgo	Fundamentos de BD
BD avanzadas	Sra. Hidalgo	Diseño de BD avanzadas
BD avanzadas	Sr. Martín	Diseño de BD avanzadas

Nom _Asignatura →→ *Profesor* *Nom _Asignatura* →→ *Texto*

Figura 6.3. Ejemplo de dependencias multivaluadas

Las dependencias multivaluadas fueron introducidas, independientemente, por ZANIOLO (1976), FAGIN (1977) y DELOBEL (1978) —aunque ya en 1973, en la tesis doctoral de este último aparece la noción— al observar las redundancias que seguían existiendo en relaciones que ya se encontraban en la última forma normal definida hasta aquel momento, la forma normal de Boyce y Codd.

2.1. Dependencias multivaluadas

De acuerdo con la definición de FAGIN (1977), una dependencia multivaluada es una sentencia:

$X \twoheadrightarrow Y$

que leemos X "multidetermina" a Y, donde X e Y son descriptores tales que un cierto valor de X implica un conjunto bien definido de valores de Y, **con independencia del resto de los atributos de la relación.**

Dada la relación R(A), se cumple:

$X \twoheadrightarrow Y$

si, para cada valor de X, hay un conjunto de cero o más valores de Y, independientemente de los valores del descriptor A - X - Y.

En el ejemplo de la figura 6.3 se cumple la definición de dependencia multivaluada, de forma que *Nom_Asignatura* multidetermina un conjunto de *Profesores*, así como a un conjunto de *Textos*, siendo los *Profesores* independientes de los *Textos*.

En cambio, la relación de la figura 6.4 no presenta la dependencia multivaluada $Cód_Curso \twoheadrightarrow Texto$, ya que, para un determinado *Cód_Curso* "A2783", el conjunto de valores del atributo *Texto* es distinto según el *Idioma*; así, para el idioma "Inglés" aparece el valor "Introducción a las BD", mientras que para el idioma "Español" aparecen "Introducción a las BD" y "Modelo Relacional". Análogamente $Cód_Curso \not\rightarrow Idioma$ porque para el *Cód_Curso* "A2783" y el *Texto* "Introducción a las BD" aparecen los idiomas {"Español", "Inglés"} mientras que para "Modelo relacional" sólo aparece "Español".

CURSOS		
<i>Cód_Curso</i>	<i>Texto</i>	<i>Idioma</i>
A2783	Introducción alas BD	Español
A2783	Introducción alas BD	Inglés
A2783	ModeloRelacional	Español
B2341	Concepción y diseño de BD	Francés
B2341	Modelorelacional	Español

$Cód_Curso \not\rightarrow Texto$
 $Cód_Curso \not\rightarrow Idioma$

Figura 6.4. Ejemplo de no existencia de dependencias multivaluadas

Podemos dar una definición formal de dependencia multivaluada de la siguiente forma:

Dado el esquema $R(A)$, la dependencia multivaluada $X \twoheadrightarrow Y$ se verificará en R si, y sólo si, para toda tupla u y v de cualquier extensión r de R tales que:

- a) $u[X] = v[X]^2$
- b) $u[Y] \neq v[Y]$
- c) $u[A - X - Y] \neq v[A - X - Y]$

existen dos tuplas t y w de r tales que:

- i) $u[X] = v[X] = t[X] = w[X]$
- ii) $t[Y] = v[Y]$ y $t[A - X - Y] = u[A - X - Y]$
- iii) $w[Y] = u[Y]$ y $w[A - X - Y] = v[A - X - Y]$

Es preciso observar que, en esta definición, no se exige que X e Y sean disjuntos, y que el significado de la misma es que, si existen las dos tuplas u y v que cumplen las condiciones a), b) y c), es decir, que tienen el mismo valor para X pero valores distintos de Y , tienen que aparecer asimismo las tuplas t y w donde se han intercambiado los valores de Y .

Observación:

En DELOBEL (1982), ULLMAN (1982) y (1988) —así como en otros autores— se puede observar que, en su definición de dependencia multivaluada, sólo imponen a las tuplas u y v la primera condición, es decir, $u[X] = v[X]$, olvidando $u[Y] \neq v[Y]$ y $u[A - X - Y] \neq v[A - X - Y]$ —aunque puedan presuponerlo no lo incluyen explícitamente—, lo que es claramente insuficiente para la definición de dependencia multivaluada como fácilmente puede comprobar el lector mediante el ejemplo de la figura 6.3. Suponiendo que las tuplas u y v son, indistintamente, las dos últimas no es necesario que aparezcan tuplas complementarias para que se cumpla la dependencia multivaluada (a pesar de que se cumpla la 1ª condición y $u[Nom_Asignatura] = v[Nom_Asignatura]$); únicamente cuando u y v son, indistintamente, la primera y la cuarta, es decir cuando se verifican las dos condiciones adicionales que nosotros hemos señalado explícitamente, es preciso que aparezcan las tuplas segunda y tercera para que se cumpla la dependencia multivaluada.

La definición anterior es más general que otra definición muy frecuente y que dice:

Dado el esquema $R(A)$, donde X e Y son descriptores y $Z = A - (X \cup Y)$, existe la dependencia multivaluada $X \twoheadrightarrow Y$ si, para cualquier valor (x, z) de los atributos X y Z , se cumple que

$$R[x, Y, z] = R[x, Y]$$

² La notación $u[X]$, $v[X]$, etc. significa la proyección de la tupla u , v , etc. sobre el correspondiente atributo —en este caso X —.

La notación anterior significa que los valores de Y asociados a (x, z) son iguales a los valores de Y asociados a x .

En esta definición es preciso exigir que X, Y, Z sean descriptores disjuntos, mientras que en la primera no se imponía tal restricción. Hemos dado, sin embargo, esta nueva definición porque, mediante ella, es muy fácil comprobar, cuando los descriptores son disjuntos, la existencia de una dependencia multivaluada en una determinada extensión³.

Las dependencias multivaluadas siempre se producen por parejas; así, si en el esquema $R(A)$ existe la dependencia $X \twoheadrightarrow Y$, al mismo tiempo habrá de cumplirse $X \twoheadrightarrow A - (X \cup Y)$, lo que se representa por:

$$X \twoheadrightarrow Y \mid A - (X \cup Y)$$

En el ejemplo de la figura 6.3 se cumple:

$$Nom_Asignatura \twoheadrightarrow Profesor \mid Texto$$

La definición de dependencia funcional es más estricta que la de dependencia multivaluada, de forma que siempre una dependencia funcional es también una dependencia multivaluada —donde se determina un único valor del implicado por cada valor del implicante—, pero la inversa no siempre es cierta, es decir, el universo de las dependencias funcionales es un subconjunto del de las multivaluadas.

A diferencia de las dependencias funcionales, donde el que se verifique $X \rightarrow Y$ depende exclusivamente de los dos descriptores involucrados en la misma, en las dependencias multivaluadas influye el resto de los atributos de la relación, es decir, las dependencias multivaluadas **dependen del contexto**. Así, si a la relación ASIGNATURAS de la figura 6.3 le añadimos el atributo *Preferencia* que nos indica el orden de preferencia de los distintos textos para cada profesor (la cual puede ser distinta para distintos profesores), obtenemos la relación de la figura 6.5, en la cual, a primera vista, parece que podría haber una dependencia multivaluada de $Nom_Asignatura \twoheadrightarrow Profesor$, pero si analizamos más detenidamente la situación, nos encontramos que:

$$ASIG_TEXTOS_PREF ["Archivos y BD", Profesor]^4 =$$

³ Al igual que ocurre con las dependencias funcionales, analizando una determinada extensión r de un esquema relacional R se puede deducir la inexistencia de una dependencia multivaluada, pero no se puede nunca "afirmar" su existencia de una manera general, únicamente podríamos decir que se cumple en **esa extensión**.

⁴ Recuerdese que la notación:

$$R [x, Y, z, \dots]$$

significa la proyección sobre Y asociada a los valores $x, z, \dots$

{ "Sr. Sánchez", "Sra. Hidalgo" }

mientras que:

ASIG_TEXTO_PREF ["Archivos y BD", *Profesor*, "1"] = { "Sr. Sánchez" }

luego no existe tal dependencia; el conjunto de valores de *Profesor* asociados a *Nom_Asignatura* depende de otros atributos de la relación (de *Pref*):

Nomb_Asignatura $\not\rightarrow$ *Profesor*⁵

ASIG_TEXTOS_PRE			
<i>Nom_Asignatura</i>	<i>Profesor</i>	<i>Texto</i>	<i>Pref</i>
Ficheros y BD	Sr. Sánchez	Concepción y Diseño de BD	1
Ficheros y BD	Sr. Sánchez	Fundamentos de BD	2
Ficheros y BD	Sra. Hidalgo	Concepción y Diseño de BD	2
Ficheros y BD	Sra. Hidalgo	Fundamentos de BD	1
BD avanzadas	Sra. Hidalgo	Diseño de BD avanzadas	1
BD avanzadas	Sr. Martín	Diseño de BD avanzadas	1

Figura 6.5. Ejemplo de relación donde no se cumple la dependencia
Nom_Asignatura $\rightarrow$ *Profesor*

Las dependencias multivaluadas serán triviales en los casos en que:

- $X = Y$
- Y es un subconjunto de X ($Y \subset X$)
- $X \cup Y = A$ (conjunto de atributos de la relación)

⁵ Más adelante veremos que las dependencias multivaluadas aparecerían en una proyección de la relación anterior, existiendo en la relación ASIG_TEXTOS_PREF una dependencia denominada embebida.

2.2. Axiomas para la derivación de dependencias funcionales y multivaluadas

Al igual que para las dependencias funcionales, también existen reglas de inferencia para las dependencias multivaluadas, las cuales incluyen los axiomas de Armstrong.

A1: Reflexividad

Si $Y \subseteq X$ entonces $X \rightarrow Y$

A2: Aumentatividad

Si $X \rightarrow Y$ y $Z \subseteq W$, entonces $XW \rightarrow YZ$

A3: Transitividad

Si $X \rightarrow Y$ e $Y \rightarrow Z$ entonces $X \rightarrow Z$

A4: Complementariedad (para dependencias multivaluadas)

Si $X \twoheadrightarrow Y$ entonces $X \twoheadrightarrow (A - (X \cup Y))$

A5: Aumentatividad (para dependencias multivaluadas)

Si $X \twoheadrightarrow Y$ y $Z \subseteq W$ entonces $XW \twoheadrightarrow YZ$

A6: Transitividad (para dependencias multivaluadas)

Si $X \twoheadrightarrow Y$ e $Y \twoheadrightarrow Z$ entonces $X \twoheadrightarrow Z - Y$

A7: Replicación

Si $X \rightarrow Y$ entonces $X \twoheadrightarrow Y$

A8: Fusión (“coalescence”)

Si $X \twoheadrightarrow Y$ y $Z \rightarrow W$, siendo $W \subseteq Y$ e $Y \cap Z = \emptyset$, entonces $X \rightarrow W$

Donde se puede observar que los tres primeros axiomas son los de Armstrong para dependencias funcionales, los tres siguientes (A4 a A6) son propios de las dependencias multivaluadas, y los dos últimos relacionan dependencias multivaluadas y funcionales.

A partir de estos axiomas se pueden derivar varios más, análogos a los existentes para las dependencias funcionales, lo que dejamos como ejercicio al lector interesado, que en cualquier caso puede encontrarlos, por ejemplo, en ULLMAN (1988) y en SILBERSCHATZ (1998).

Este conjunto de axiomas es correcto y completo. Es correcto en el sentido de que, dado $R(A, D)$ donde D es un conjunto de dependencias —funcionales y multivaluadas—, toda dependencia que se deduzca por aplicación de estas reglas se satisface para toda extensión r del esquema R . Es completo porque, a partir de este conjunto de axiomas, se puede deducir toda dependencia que se cumpla para R —la demostración se encuentra en BEERI (1977)—.

Al igual que para las dependencias funcionales, también la aplicación de estos axiomas permite el cálculo del cierre D^+ de un conjunto D de dependencias —funcionales y multivaluadas— que será el conjunto D de dependencias de D más todas las que se puedan deducir de ellas por aplicación de los anteriores axiomas. Por tanto, si se desea conocer si $X \twoheadrightarrow Y$ viene implicada por D (lo que se expresa, al igual que para las dependencias funcionales, por $D \models X \twoheadrightarrow Y$) habría que calcular D^+ y comprobar si $X \twoheadrightarrow Y \in D^+$. Sin embargo el cálculo de D^+ tiene un elevado coste computacional, dado que es exponencial con el número de dependencias, por lo que BEERI (1980) propone calcular lo que llama "base de dependencias" para el implicante X , y demuestra que $X \twoheadrightarrow Y$ si $X - Y$ es la unión de algunos de los descriptores de la base. Apoyándose en estas conclusiones BEERI (1980) da un algoritmo con un tiempo polinomial, que aparece también en ULLMAN (1988).

2.3. Cuarta forma normal (4FN)

Diremos que una relación se encuentra en 4FN si, y sólo si, las únicas dependencias multivaluadas no triviales son aquellas en las cuales una clave multi-determina un atributo, es decir, toda dependencia multivaluada viene determinada por una clave candidata.

En el ejemplo que hemos visto de:

ASIGNATURAS (Nom_Asignatura, Profesor, Texto)

donde existen las dependencias multivaluadas:

Nom_Asignatura $\twoheadrightarrow$ Profesor

Nom_Asignatura $\twoheadrightarrow$ Texto

la relación ASIGNATURAS no se encuentra en 4FN, ya que estas dependencias multivaluadas están implicadas por *Nom_Asignatura*, que no es una clave candidata; la clave candidata es el conjunto de los tres atributos (*Nom_Asignatura*, *Profesor*, *Texto*).

Para evitar las anomalías que, por no estar en 4FN, se pueden producir al actualizar dicha relación, es preciso descomponerla en dos proyecciones, quedando:

ASIGNATURAS1 (*Nom_Asignatura*, *Profesor*)

ASIGNATURAS2 (*Nom_Asignatura*, *Texto*)

que sí están en 4FN.

La descomposición sin pérdida de información para el caso de dependencias funcionales tiene su equivalencia cuando se trata de dependencias multivaluadas:

Dado el esquema de relación:

R (A, D)

donde D es un conjunto de dependencias (funcionales y multivaluadas), las proyecciones R1 y R2 cumplen la propiedad de descomposición sin pérdida de R, si se cumple:

$$R1 \cap R2 \rightarrow\rightarrow R1 - R2$$

también:

$$R1 \cap R2 \rightarrow\rightarrow R2 - R1$$

La descomposición de la relación ASIGNATURAS en ASIGNATURAS1 y ASIGNATURAS2 ha sido realizada sin pérdida de información como puede fácilmente comprobar el lector.

En FAGIN (1977) se puede encontrar la demostración de que una relación R (A, B, C) se puede descomponer sin pérdida en sus dos proyecciones R1 (A, B) y R2 (A, C), si, y sólo si, existe en R la dependencia multivaluada $A \rightarrow\rightarrow B \mid C$.

Cuando se aborda en la práctica el proceso de normalización por descomposición, es muy frecuente comenzar por las dependencias multivaluadas y descomponer primero en proyecciones independientes teniendo en cuenta este tipo de dependencias; después, se continuará la normalización de dichas proyecciones que pueden no

encontrarse en 2FN, 3FN o FNBC. Aunque es preciso tener en cuenta que al descomponer en base a las dependencias multivaluadas se pueden perder dependencias funcionales.

3. DEPENDENCIAS MULTIVALUADAS EMBEBIDAS

Es un tipo de restricción que posee la propiedad de ser multivaluada sobre una proyección de una relación (obsérvese que esto nunca puede ocurrir con una dependencia funcional dado que en este tipo de dependencia no influye el contexto, sino únicamente los atributos implicados en la dependencia).

Si volvemos al ejemplo de la figura 6.5, en ella no existían dependencias multivaluadas, a pesar de lo que a primera vista pudiera parecer; sin embargo, si proyectamos tal relación sobre (*Nom_Asignatura*, *Profesor*, y *Texto*), tenemos la relación de la figura 6.3 donde existían las dependencias multivaluadas

Nom_Asignatura $\rightarrow\rightarrow$ *Profesor* | *Texto*

Este tipo de dependencias multivaluadas que no existen en un cierto esquema de relación y que sólo aparecen en una proyección de la relación original, se denominan "embebidas" —FAGIN (1977), ULLMAN (1988)— o también "jerárquicas" —DELOBEL (1982)—.

Una relación $R(A)$ satisface una dependencia multivaluada embebida $X \rightarrow\rightarrow Y \mid Z$, donde $X \cup Y \cup Z \subset A$, si dicha dependencia multivaluada no existe en R , cumpliéndose, en cambio, para cualquier extensión del esquema $R' (X \cup Y \cup Z)$ que es una proyección de R sobre los atributos $X \cup Y \cup Z$ involucrados en la dependencia embebida (los descriptores X, Y, Z no tienen por qué ser disjuntos).

Siguiendo a DELOBEL (1982) daremos la siguiente definición más general de dependencia jerárquica de orden K .

Sea una relación $R(A)$, en la cual puede distinguirse un conjunto de descriptores disjuntos $X, Y_1, Y_2, \dots, Y_k$ y W , cuya unión es igual a A , se dice que existe una dependencia jerárquica de orden k , que denotamos

$X: Y_1 \mid Y_2 \mid \dots \mid Y_k$

si se cumple que, para todo valor $x \in X$

$R[x, Y_1, Y_2, \dots, Y_k] = R[x, Y_1] * R[x, Y_2] * \dots * R[x, Y_k]$

Las dependencias jerárquicas se pueden ver como dependencias que generalizan las multivaluadas, siendo estas últimas un subconjunto de las jerárquicas. Una dependencia jerárquica será también multivaluada cuando es de orden 2 y el componente W es el conjunto vacío:

$$X \text{ U } Y_1 \text{ U } Y_2 = A, \text{ por lo que } X \rightarrow\rightarrow Y_1 \mid Y_2$$

La figura 6.6, que refleja las notas que ponen los profesores a los alumnos en distintas asignaturas, es otro ejemplo de dependencias multivaluadas embebidas.

CALIFICACIONES			
<i>Materia</i>	<i>Profesor</i>	<i>Alumno</i>	<i>Nota</i>
BD	Sr. Sánchez	Javier	8
BD	Sr. López	Javier	6
BD	Sr. Sánchez	Susana	9
BD	Sr. López	Susana	8
BD			...
....			...
SI	Sr. García	Javier	9

Figura 6.6. Ejemplo de dependencias jerárquicas o multivaluadas embebidas

En la relación CALIFICACIONES la única dependencia funcional o multivaluada, no trivial, existente es:

$$\text{Profesor, Alumno} \rightarrow \text{Nota}$$

siendo la clave de la relación CALIFICACIONES (*Materia, Profesor, Alumno*), luego esta relación no está en segunda forma normal, ya que *NOTA* viene implicada por (*Profesor, Alumno*) que no es clave, por lo que habría que descomponer en:

CALIF1 (*Profesor, Alumno, Nota*)

CALIF2 (*Materia, Profesor, Alumno*)

En esta segunda relación aparecen ya las dependencias multivaluadas

$Materia \twoheadrightarrow \twoheadrightarrow Profesor \mid Alumno$

Que, al no estar en 4FN, habría que descomponer en dos proyecciones independientes, quedando el siguiente esquema relacional:

CALIF1 (*Profesor, Alumno, Nota*)

CALIF2.1 (*Materia, Profesor*)

CALIF2.2 (*Materia, Alumno*)

En este ejemplo la relación no estaba en 2FN y se podía descomponer en dos proyecciones independientes, en una de las cuales aparecían las dependencias multivaluadas. Existen, sin embargo, otros casos en los que la relación está en 4FN y, a pesar de ello, existen dependencias multivaluadas embebidas; veamos un ejemplo:

Supongamos la misma relación de la figura 6.6 pero con una semántica distinta, ya que, en lugar de la anterior dependencia funcional $Profesor, Alumno \rightarrow Nota$, tenemos ahora la siguiente:

$Materia, Profesor, Alumno \rightarrow Nota$

por lo que la relación CALIFICACIONES tiene como clave:

$\{ Materia, Profesor, Alumno \}$

y su única dependencia funcional viene implicada por la clave, estando por tanto en FNBC. En la relación no hay dependencias multivaluadas, como es muy fácil comprobar; sin embargo, existen redundancias y las consiguientes anomalías. Si proyectáramos sobre $\{Materia, Profesor, Alumno\}$ aparecerían las dependencias multivaluadas

$Materia \twoheadrightarrow \twoheadrightarrow Profesor \mid Alumno$

Pero no es posible descomponer CALIFICACIONES en proyecciones independientes, es decir, no se puede conseguir una descomposición sin pérdida.

La relación tiene dependencias multivaluadas embebidas, pero cumple la definición de 4FN —también la de 5FN— al no tener dependencias multivaluadas.

Al igual que para las dependencias funcionales y multivaluadas, para las dependencias jerárquicas existen también unas reglas de derivación que permiten obtener nuevas dependencias funcionales, multivaluadas y jerárquicas; pero, a diferencia de los casos anteriores, no se puede garantizar la obtención del cierre de todas las posibles dependencias; ya que, como se demuestra en PARKER (1980), no existe un conjunto finito y completo de axiomas para las dependencias jerárquicas.

4. DEPENDENCIAS DE COMBINACION Y QUINTA FORMA NORMAL

Las dependencias funcionales y multivaluadas analizadas hasta ahora permiten la descomposición sin pérdida de una relación en dos de sus proyecciones; sin embargo, existen relaciones en las que no se puede llevar a cabo tal descomposición binaria sin pérdida de información, como en el ejemplo que proponemos en la figura 6.7, donde se muestra la relación EDITA, que nos indica que una editorial publica en un determinado idioma sobre ciertos temas.

EDITA		
<i>Editorial</i>	<i>Idioma</i>	<i>Tema</i>
RA-MA	Inglés	BD
RA-MA	Español	CASE
Addison Wesley	Español	BD
RA-MA	Español	BD

Figura 6.7. Dependencia de combinación

Aun no existiendo en esta relación ninguna dependencia funcional, ni multivaluada —tampoco embebida—, se puede observar que existen en ella redundancias y sus correspondientes anomalías, producidas por un nuevo tipo de dependencias, las **dependencias de combinación**; estas anomalías se ponen de manifiesto cuando se actualiza la base de datos, como demostraremos en los siguientes ejemplos:

Si se deseara modificar el tema "Bases de Datos" en la última tupla de EDITA habría que cambiarlo también en, al menos, una de las otras tuplas, ya que es forzoso que aparezca la tupla

< RA-MA, Español, Bases de Datos >

Si tuviéramos una extensión con sólo las dos primeras tuplas y quisiésemos insertar la tercera

< Addison Wesley, Español, Bases de Datos >

deberíamos insertar también la cuarta.

El borrado de una tupla —la cuarta— tiene efectos secundarios, ya que no se puede borrar, sin eliminar también una de las otras.

Todas estas anomalías son debidas a que los atributos de la relación EDITA no son independientes, sino que existen, entre ellos, unas ciertas restricciones que podemos expresar de la siguiente forma:

Si una editorial (RA-MA) publica sobre un determinado tema (Bases de Datos); si, además, esa editorial (RA-MA) publica en un cierto idioma (Español); y si se está publicando —en este caso Addison Wesley— sobre ese tema (Bases de Datos) en el idioma (Español).

Forzosamente tiene que ocurrir que:

La editorial (RA-MA) ha de publicar sobre el tema "Bases de Datos" en el idioma "Español"

es decir, si existen las tres tuplas indicadas, también aparecerá, obligatoriamente, la tupla

< RA-MA, Español, Bases de Datos >

Si descomponemos esta relación EDITA en dos de sus proyecciones, en la combinación de las mismas aparecerá una tupla que no estaba en la relación original, es la tupla "espuria":

< Addison Wesley, Español, CASE >

Lo que nos muestra que existen relaciones que no se pueden descomponer en dos proyecciones sin pérdida de información.

En cambio, si la descomposición se lleva a cabo en tres relaciones, como se muestra en la figura 6.8, la combinación de ellas sí nos devuelve la relación original:

$$\text{EDITA} = \text{EDITA1} * \text{EDITA2} * \text{EDITA3}$$

habiéndose efectuado la descomposición **sin pérdida de información**.

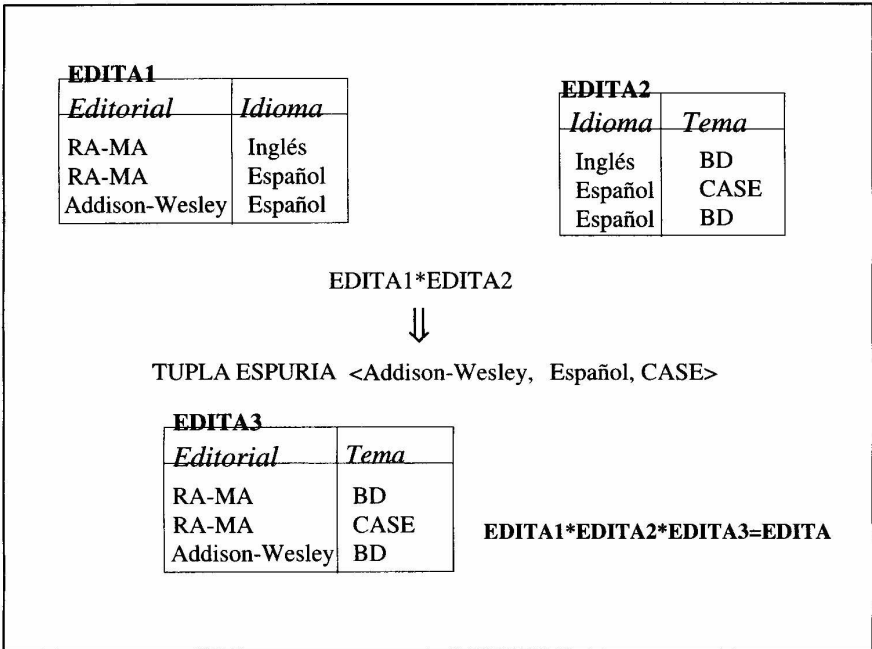


Figura 6.8. Descomposición de la relación EDITA de la figura anterior en proyecciones independientes

Por tanto, la relación EDITA tiene una restricción, que se ha de cumplir para todas sus extensiones (es independiente del tiempo), a la que se denomina **dependencia de combinación**.

El concepto de dependencia de combinación fue formalizado por RISSANEN (1979) en un trabajo que extendía las conclusiones de NICOLAS (1978) sobre lo que éste denominó dependencias "mutuas". Las dependencias de combinación son, al igual que las otras dependencias, una restricción sobre una relación y constituyen una generalización de las anteriores, de forma que una dependencia funcional es siempre también multivaluada y de combinación, y una multivaluada es también de combinación, pero la afirmación inversa no es cierta.

El descubrimiento de esta nueva forma de dependencia llevó a definir también una nueva forma normal, la quinta (5FN) que, como las anteriores, significa un paso adelante en la eliminación de redundancias.

La relación EDITA del ejemplo anterior, está en 4FN, al no tener dependencias funcionales ni multivaluadas, pero aún subsisten en ella redundancias; la razón es que no se encuentra en 5FN.

La dependencia de combinación está asociada a la 5FN —que se llama también de "proyección/combinación"⁶—, que es la última forma normal asociada a estos operadores.

4.1. Definición de dependencia de combinación

Una relación se dice que tiene la propiedad de "**dependencia de combinación**", respecto de sus proyecciones $R_1, R_2, \dots, R_j$, si:

$R = R_1 * R_2 * \dots * R_j$, y se denota por:

$DJ * (R_1, \dots, R_j)$

Una dependencia multivaluada es un caso especial de una dependencia de combinación en la que $R_j = Z$.

Una dependencia de combinación es trivial si cualquiera de los esquemas de relación R_i es igual a R .

Si R presenta la propiedad de dependencia de combinación respecto de R_1, R_2 y R_3 , entonces $\langle A_1, B_1, C_1 \rangle \in R$ si y sólo si $\langle A_1, B_1 \rangle \in R_1, \langle B_1, C_1 \rangle \in R_2$ y $\langle C_1, A_1 \rangle \in R_3$. O sea, no pueden aparecer en R solas las tuplas $\langle A_1, B_1, C_2 \rangle, \langle A_2, B_1, C_1 \rangle$ y $\langle A_1, B_2, C_1 \rangle$ por ejemplo, sino que debe aparecer junto a ellas la tupla adicional $\langle A_1, B_1, C_1 \rangle$.

4.2. Quinta forma normal (5FN)

Decimos que una relación R está en 5FN si, y sólo si, está en todas las anteriores formas normales y toda dependencia de combinación está implicada por una clave candidata.

⁶ En inglés, *Projection-Join Normal Form (PJ/NF)*

Al igual que ocurre con la FNBC y con la 4FN, podríamos también dar otra definición más sencilla de 5FN, sin apoyarnos en las anteriores formas normales, diciendo que "una relación está en 5FN si, y sólo si, toda dependencia funcional, multivaluada o de combinación no trivial es consecuencia de las claves candidatas".

Una relación que no se encuentre en 5FN por tener una dependencia de combinación sobre sus descriptores $X_1, X_2, \dots, X_j$ puede ser descompuesta **sin pérdida de información** en j proyecciones independientes, es decir:

$$R = R_1(X_1) * R_2(X_2) * \dots * R_j(X_j)$$

esta propiedad es análoga a la descomposición sin pérdida de información que estudiamos para las dependencias funcionales y multivaluadas.

En el ejemplo de la figura 6.8 ya hemos visto que su descomposición correcta de EDITA, al existir una dependencia de combinación, es la que aparece en dicha figura.

Al igual que en las dependencias multivaluadas, también pueden existir dependencias de combinación embebidas que no se cumplen en la relación original y sólo aparecen en determinadas proyecciones de la misma.

4.3. Dependencia de dominio/clave

En FAGIN (1981) se propone la forma normal de dominio/clave (FNDC), demostrándose que toda relación que se encuentra en FNDC se encuentra necesariamente en 5FN.

Para definir esta forma normal, Fagin no utiliza ninguna de las dependencias que hemos expuesto anteriormente, afirmando que: "Una relación R se encuentra en FNDC si, y sólo si, toda restricción de R es una consecuencia lógica de las restricciones de dominio y las restricciones de clave que se cumplen en R ".

Como señala DATE (1990), aunque llevar una relación a FNDC es conceptualmente muy sencillo —basta con hacer cumplir las restricciones de dominio y las de clave—, no se ha demostrado si esta forma normal se puede alcanzar para todas las relaciones ni bajo qué condiciones se puede lograr.

5. OTRAS DEPENDENCIAS Y FORMAS NORMALES

Durante la década de los ochenta se fueron proponiendo otros tipos de dependencias, además de las expuestas en este capítulo y en el anterior; aunque en

general han tenido poca influencia práctica en la teoría del diseño de bases de datos. Una excepción es la de las dependencias de inclusión, que son una generalización del concepto de integridad referencial, y sirven para llevar al modelo relacional conceptos semánticos como el de interrelación y generalización ("ES_UN") del modelo E/R.

Otras dependencias a destacar son las denominadas "dependencias de plantilla" ("*template dependencies*") y las dependencias algebraicas, que el lector interesado puede encontrar expuestas en ULLMAN (1988) y GARDARIN y VALDURIEZ (1990).

5.1. Dependencias de inclusión

Todas las dependencias anteriormente expuestas se definían sobre los atributos de un mismo esquema de relación, pero existen otro tipo de dependencias que involucran atributos de esquemas de relación distintos, son las **dependencias de inclusión**.

Decimos que el descriptor X del esquema de relación R depende en inclusión del descriptor Y del esquema de relación S, lo que se denota por:

$$R[X] \subseteq S[Y]$$

si para cualquier extensión r de R y s de S, se cumple:

$$\Pi_X(r) \subseteq \Pi_Y(s)$$

siendo X e Y de la misma longitud y definidos sobre los mismos dominios (R y S no necesariamente distintos).

En CASANOVA *et al.* (1982) se definen reglas de inferencia para deducir dependencias de inclusión a partir de otras.

En MANNILA y RAIHA (1986) se define la "Forma Normal de Dependencias de Inclusión" (FNDI) de la siguiente forma:

Se dice que un esquema relacional con dependencias funcionales y de inclusión está en FNDI si:

- Los esquemas de relación están en 3FN
- Las dependencias de inclusión están basadas en claves
- El conjunto de dependencias de inclusión es no cíclico

A continuación, exponemos con más detalle estas dos últimas condiciones.

Se dice que una dependencia de inclusión está basada en claves si, en la definición anterior de dependencia de inclusión, Y es una clave de S .

Por otra parte, se dice que un conjunto de dependencias de inclusión es cíclico, SCIORE (1983), si se cumple cualquiera de las condiciones siguientes:

- i) Existe un esquema relacional R y dos descriptores X e Y tales que $R[X] = R[Y]$
- ii) Existen los esquemas de relación $R_1, \dots, R_n$ ($n > 1$) tales que:
 $R_1[X_1] \subseteq R_2[X_2], R_2[X_2] \subseteq R_3[X_3], \dots, R_n[X_n] \subseteq R_1[X_1]$

6. OTRAS CONSIDERACIONES SOBRE LA NORMALIZACIÓN DE RELACIONES

En la metodología de diseño de bases de datos que proponemos en esta obra, el diseño lógico de una BD comienza aplicando un conjunto de reglas de derivación al esquema conceptual en el ME/R (obtenido en una fase anterior) para transformarlo en un esquema relacional de partida que va sufriendo un proceso de refinamientos sucesivos mediante la aplicación de la teoría de la normalización, la cual tiene por objeto "depurar" la presencia de datos redundantes y de posibles incoherencias, cuyo origen es una defectuosa percepción del mundo real, una inadecuada representación en el ME/R, o una aplicación incorrecta de las reglas de derivación del esquema relacional —a veces por las tres causas—.

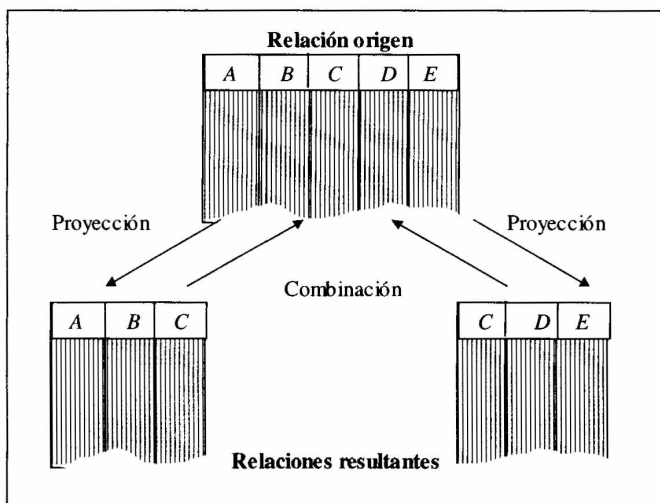


Figura 6.9. Proceso de normalización de relaciones por descomposición

Mediante el método de análisis, vamos —según se ha expuesto— transformando, en pasos sucesivos, las relaciones que presentan ciertos problemas —por ejemplo anomalías de inserción, borrado y modificación— en otras de menor grado utilizando el operador de proyección; pasando, así, de relaciones que se encuentran en primera forma normal a otras que están en formas normales más avanzadas. Siempre ha de ser posible reconstruir, mediante la aplicación del operador de combinación ("join"), la relación origen a partir de las relaciones resultantes —figura 6.9—.

Nosotros propugnamos llegar hasta la 5FN, aunque posteriormente sea preciso (a menudo por razones de eficiencia) retroceder hasta formas anteriores⁷.

La distribución aproximada de los distintos tipos de dependencias en aplicaciones prácticas se muestran en la figura 6.10, donde se puede observar la escasa proporción de dependencias de combinación "puras"⁸, en el mundo real, respecto al resto de dependencias (funcionales y multivaluadas); también el número de dependencias multivaluadas "puras" es bastante menor que el de las funcionales. No hay, por tanto, muchas relaciones que encontrándose en FNBC no estén también en 4FN y menos aún las que estando en 4FN no estén también en 5FN⁹.

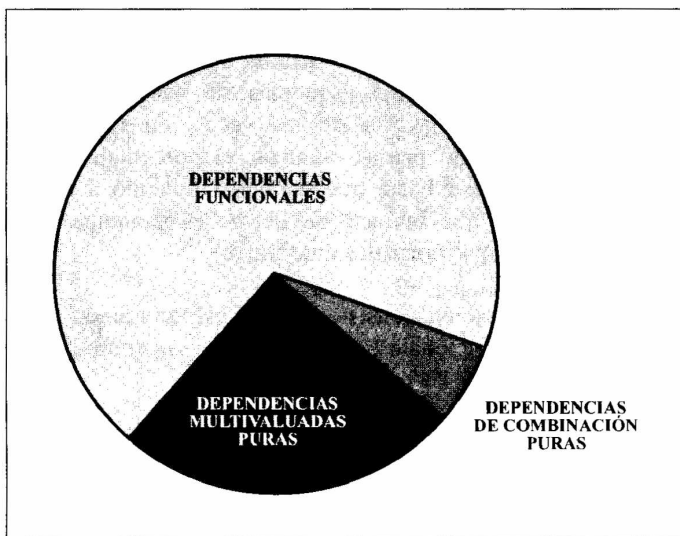


Figura 6.10. Distribución aproximada de las dependencias en aplicaciones prácticas, según DELOBEL y ADIBA (1982)

⁷ Muchos autores, por razones prácticas, aconsejan no proseguir el proceso de normalización más allá de la 3FN.

⁸ Con la expresión de dependencias de combinación "puras" significamos aquellas que son *sólo* de combinación, pero no multivaluadas ni funcionales. Análogo es el significado de multivaluadas "puras".

⁹ En nuestra opinión, si se sigue la metodología de diseño propuesta, estos casos se presentan casi siempre por haber realizado mal el diseño en el ME/R

La principal dificultad para llegar hasta 5FN no se halla, por otra parte, en el propio proceso de normalización, sino en la detección de las dependencias (en particular de las de combinación, especialmente en esquemas con gran cantidad de atributos), por lo que si las dependencias han podido ser determinadas, el normalizar hasta la 5FN no supone demasiado esfuerzo y, en cambio, preserva el rigor metodológico, llevando a un estudio más profundo de las relaciones.

Éstas son las principales razones que nos hacen proponer (siempre que no se pierdan dependencias funcionales) una normalización hasta 5FN, aun cuando sea necesario un posterior proceso de desnormalización; de esta forma seremos conscientes de las razones que nos han llevado a desnormalizar, debiendo quedar en la documentación el esquema en 5FN.

Además, si se sigue una metodología que parta de un esquema E/R y se aplica correctamente un conjunto de reglas de derivación, se llega, salvo contadas excepciones, directamente a un esquema relacional en 5FN.

Esta insistencia en aconsejar que se avance lo más posible en la normalización (entre otras razones por lo que supone de profundización en la comprensión de nuestro universo del discurso al detectar y analizar las dependencias) no es incompatible con nuestro reconocimiento de que, en el estado actual de la técnica y ante las exigencias de eficiencia de las aplicaciones, los esquemas de bases de datos no pueden mantenerse en la práctica, la mayoría de las veces, en formas normales muy avanzadas. Además, es asimismo preciso analizar el porcentaje de actualizaciones frente al de consultas para saber hasta qué punto, atendiendo a unas determinadas aplicaciones, es conveniente que la base de datos se mantenga en 5FN o si es preferible retroceder hasta formas normales anteriores.

La teoría de la normalización consiste en una estructuración de las relaciones basada en las dependencias (funcionales, multivaluadas y de combinación), pero no es ésta la única forma de estructurar las relaciones.

En el próximo epígrafe expondremos las técnicas que nos permiten la reorganización de relaciones, atendiendo a consideraciones de tipo lógico y a consideraciones de tipo físico.

7. REORGANIZACIÓN DE RELACIONES

El desarrollo de una base de datos suele descomponerse en tres grandes fases —modelado conceptual, diseño lógico y diseño físico—, lo que permite reducir la complejidad que entraña el diseño, a la vez que ayuda a alcanzar los dos principales objetivos que, según HAINAUT (1989), tienen las bases de datos:

- Ser una representación fidedigna del mundo real
- Ser un "servidor" operacional y eficiente de los datos

El modelo relacional es el único modelo que ha permitido abordar la fase de diseño lógico aplicando una teoría formal para la estructuración de las relaciones: el proceso de normalización.

Sin embargo, la normalización se muestra insuficiente para alcanzar los objetivos de diseño lógico y físico, por lo que, en la práctica, es preciso muchas veces proceder a otro tipo de organización de las relaciones. Esta organización se puede descomponer en dos subetapas: la primera consiste en la estructuración de las relaciones atendiendo a consideraciones de tipo lógico, donde se realiza la normalización así como un particionamiento horizontal si fuese necesario, y en la segunda se reestructuran las relaciones por consideraciones de tipo físico, lo que puede llevar a la desnormalización, al particionamiento horizontal o al particionamiento vertical —véase figura 6.11—. En nuestra opinión, la razón de esta etapa de reestructuración se encuentra en la falta de flexibilidad de la estructura interna de los actuales SGBD, los cuales no ofrecen los adecuados instrumentos de diseño físico, obligando así a realizar cambios en la estructura lógica de las relaciones por consideraciones de eficiencia.

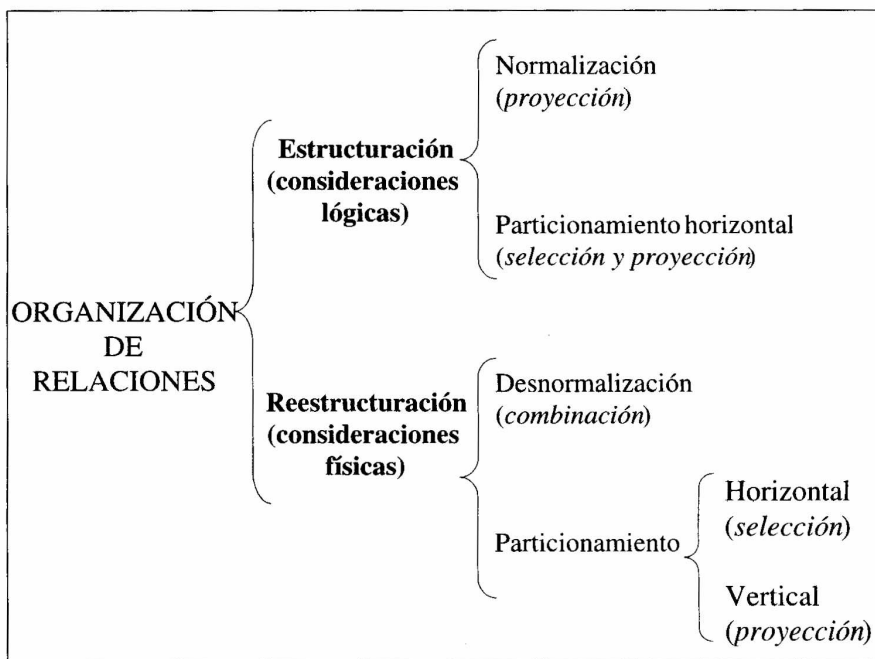


Figura 6.11. Estructuración y reestructuración de relaciones

7.1. Estructuración de relaciones por consideraciones lógicas: particionamiento horizontal

Además de la normalización, existe otro tipo de estructuración de relaciones atendiendo a requisitos lógicos: el **particionamiento horizontal**, donde se aplica el operador de selección, seguido de proyecciones sobre las relaciones resultantes (véase figura 6.12). Esta estructuración permite suprimir valores nulos —inaplicables— que pueden aparecer en las relaciones debido, en general, a no haberse detectado los subtipos de una entidad, o a haberlos reunido en una única relación en el paso al modelo relacional.

Así, por ejemplo, si tuviésemos la relación:

DOCUMENTOS (*Cód_Documento, Título, Idioma, Editorial*)

que almacena los datos relativos a libros y artículos; al ser el atributo EDITORIAL inaplicable para los artículos, podríamos descomponer esta relación, con el fin de evitar valores nulos, de la siguiente forma:

LIBROS (*Cód_Documento, Título, Idioma, Editorial*)

ARTICULOS (*Cód_Documento, Título, Idioma*)

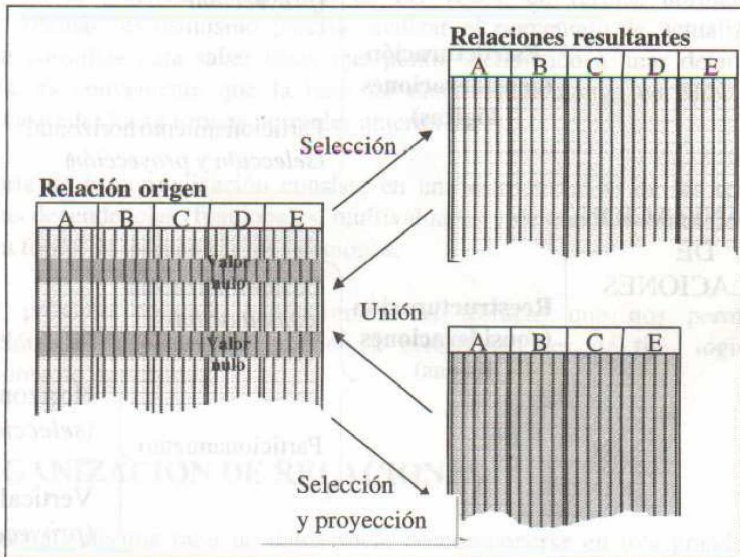


Figura 6.12. Particionamiento horizontal de relaciones

Como puede observarse, el particionamiento se ha realizado mediante el operador de selección (restricción) del álgebra relacional que divide la relación original en dos, dependiendo de que las tuplas cumplan o no un determinado predicado; (valor nulo en uno o en varios atributos) la selección va seguida, en las relaciones resultantes que contienen tuplas con atributos inaplicables¹⁰ de una proyección sobre los atributos cuyos valores no son inaplicables (esto es lo que ha ocurrido en ARTÍCULOS), donde ha desaparecido el atributo *Editorial*.

En resumen, en la primera etapa de estructuración de relaciones por razones lógicas, hemos propugnado avanzar hasta la 5FN en el proceso de normalización, así como eliminar los valores nulos —inaplicables— mediante un particionamiento horizontal basado en selección seguido de proyección¹¹. La reconstrucción de la relación original se realiza por medio del operador "unión", después de haber igualado el contexto de las dos relaciones, es decir, después de añadir los atributos que hubiesen desaparecido en la proyección. Esto no quiere decir que las relaciones que vayamos a almacenar en la base de datos sean las que resultan de estos procesos, ya que hemos de tener presente el segundo de los objetivos citados anteriormente: el de que la base de datos ha de ser un "servidor operacional y eficiente de los datos", por lo que debe existir una segunda etapa consistente en la reestructuración de relaciones cuyo objetivo principal sea mejorar la eficiencia de la base de datos. Aun cuando pudiera parecer que en el proceso de estructuración de relaciones por motivos lógicos, se trata de hacer algo que posteriormente ha de deshacerse, ya hemos explicado las razones metodológicas y prácticas que existen para ello.

Como se muestra en la figura 6.13, una vez estructurado el esquema origen, analizando las distintas dependencias y la posibilidad de valores inaplicables, se obtiene un esquema relacional en 5FN al que se añaden las claves ajenas y otras restricciones de integridad. A partir de este esquema lógico estructurado, y después de examinados los requisitos de determinadas vistas o aplicaciones críticas, se empieza un proceso de reestructuración donde priman las consideraciones de eficiencia.

¹⁰ Una sola relación en nuestro ejemplo, pero, en general, puede ser en dos o más.

¹¹ Si el modelado conceptual y la transformación al modelo relacional se hubiera realizado de forma adecuada, la estructuración de las relaciones no sería necesaria. La necesidad del particionamiento horizontal por razones lógicas surge por no haber detectado, en la fase de modelado conceptual, jerarquías o por haber unido, en la transformación al modelo relacional, tipos y/o subtipos de una jerarquía.

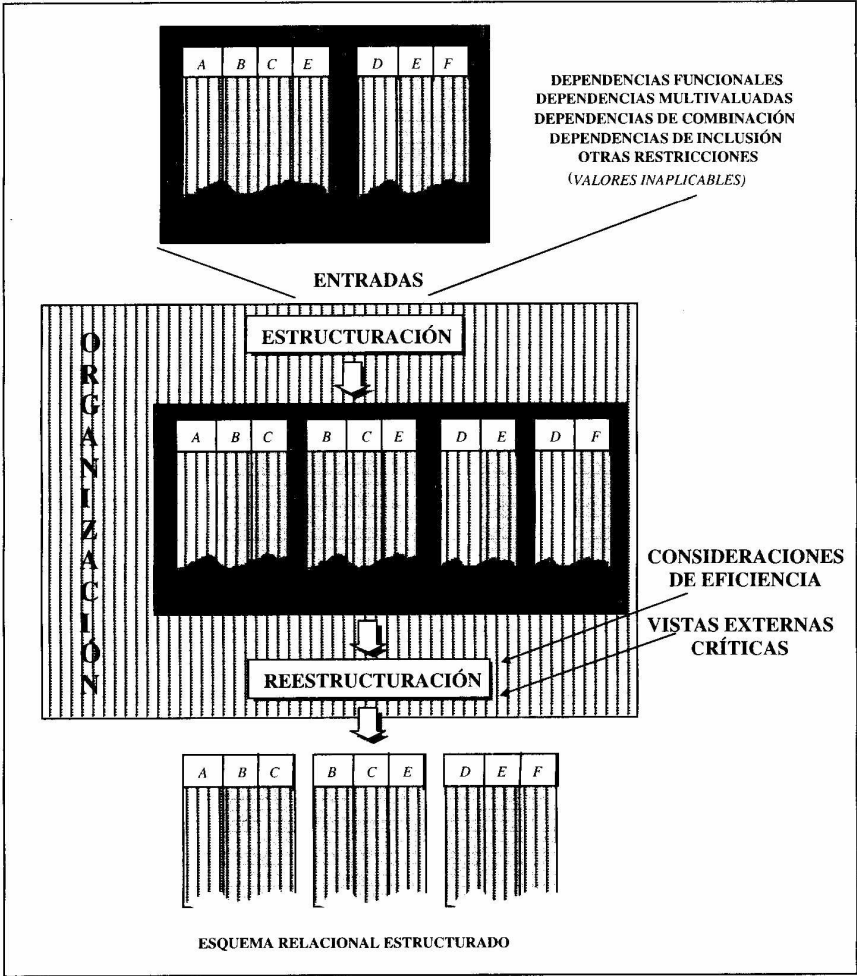


Figura 6.13. Proceso de estructuración/reestructuración

7.2. Reestructuración de relaciones por consideraciones de eficiencia: desnormalización y particionamiento

Tanto la desnormalización como el particionamiento de relaciones son, al igual que la normalización y el particionamiento horizontal para eliminar los valores inaplicables, métodos o formas de organizar los datos; pero en la reestructuración de las relaciones se tienen en cuenta consideraciones de tipo físico, como son:

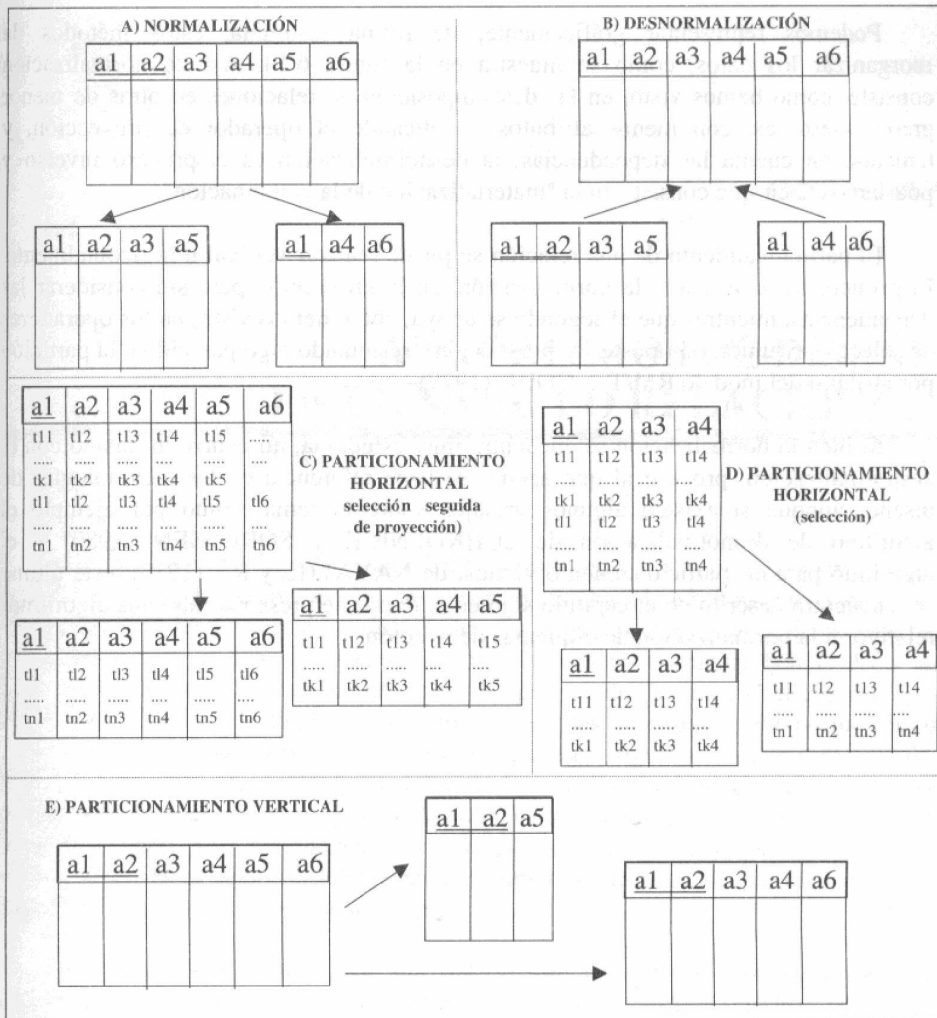


Figura 6.14. Distintas formas de organizar los datos en relaciones

- La tasa de actualizaciones respecto a la de recuperaciones
- Las veces que se accede conjuntamente a los atributos
- La longitud de los mismos
- El tipo de proceso (en línea/por lotes)
- La prioridad de los procesos
- El tamaño de las tablas
- Etc.

Podemos representar gráficamente, de forma resumida, estos métodos de reorganizar los datos, como se muestra en la figura 6.14. Así, la normalización consiste, como hemos visto, en la descomposición de relaciones en otras de menor grado —esto es, con menos atributos— aplicando el operador de proyección, y teniendo en cuenta las dependencias; la desnormalización es el proceso inverso y podríamos decir que consiste en la "materialización de la combinación".

El particionamiento de una relación se puede realizar vertical u horizontalmente. El primero se basa, como la normalización, en la proyección pero sin considerar las dependencias, mientras que el segundo se apoya, como hemos visto, en los operadores de selección y unión (sin posterior proyección), resultando algo parecido a la partición por atributo del modelo RM/T —Codd (1979)—.

Si bien la normalización se encuentra muy estudiada, no ocurre lo mismo con la desnormalización, proceso al que apenas se hace referencia en las metodologías de diseño, aunque sí existen algunos trabajos sobre el tema, como por ejemplo el algoritmo de desnormalización de SCHKOLNICK y SORENSEN (1980) y el algoritmo para el particionamiento vertical de NAVATHE y RA (1989); este último se encuentra descrito en el capítulo siguiente, donde se presentan algunos algoritmos relativos a la normalización de esquemas de relación.

ALGORITMOS DE DISEÑO EN EL MODELO RELACIONAL¹

Una de las ventajas del tratamiento formal que constituye la teoría de la normalización es la posibilidad de crear procedimientos algorítmicos que faciliten el diseño de bases de datos relacionales. En los dos capítulos anteriores se ha expuesto la teoría del diseño lógico de bases de datos relacionales y, aunque hemos dado las definiciones de los distintos conceptos y hemos presentado los procesos necesarios para obtenerlos, no hemos detallado los algoritmos asociados. La razón de ello es que hemos considerado más conveniente para el lector agrupar estos algoritmos en un solo capítulo, donde las personas interesadas (especialmente los estudiantes cuando se use el libro en ámbitos académicos) puedan encontrar los principales algoritmos relativos a la teoría de la normalización, mientras que aquellos lectores que buscan un enfoque más práctico y menos formal no se vean constantemente interrumpidos en su lectura por la inclusión de algoritmos cuyo estudio no desean abordar.

Son muchos los autores que han tratado este tema del diseño algorítmico de esquemas relacionales, entre otros, BERSTEIN (1976), BEERI (1979), ULLMAN (1982), MAIER (1983), CERI y GOTTLOB (1986), etc. Nosotros nos hemos basado principalmente en DIEDERICH y MILTON (1988), MANILA (1992), donde se proponen nuevos métodos y algoritmos más rápidos para la normalización de bases de datos.

¹ Este capítulo forma parte de los trabajos de fin de carrera de la Facultad de Informática de la UPM realizados por Henar Pinilla y Visitación López, así como de la EPS de la Universidad Carlos III de Madrid llevado a cabo por Julia Martínez. Todos ellos dirigidos por la coautora de este libro, Adoración de Miguel. Queremos agradecer a las autoras de estos proyectos su autorización para incluir este material en la presente obra. Se han introducido algunos cambios y se ha añadido un algoritmo de particionamiento vertical.

Estos algoritmos, a excepción del de particionamiento vertical, se han programado en PROLOG y se ha desarrollado una interfaz de usuario en DELPHI². Se incluye con el libro un disquete con una versión beta de esta aplicación, a la que hemos denominado RENO (**Relaciones Normalizadas**)³.

1. INTRODUCCIÓN

Los algoritmos de normalización son herramientas de ayuda al diseño de base de datos relacionales y resultan especialmente útiles cuando se parte de la relación universal y se desea obtener esquemas normalizados. Originariamente surgieron como complemento a la teoría matemática del modelo relacional y como solución para obtener esquemas normalizados de una forma automática. Según se han ido extendiendo los SGBD relacionales, así como con la introducción de herramientas CASE, se ha ido viendo la posible utilización práctica de dichos algoritmos, formando parte, a veces, de productos CASE. Sin embargo, esta utilización práctica se ha visto a veces frenada por el elevado coste computacional de estos algoritmos, lo que los hace inviables en bases de datos reales que tienen un gran número de dependencias.

Existen dos tipos de algoritmos de normalización: **descomposición** (análisis) y **síntesis**. El primero de ellos se basa en la descomposición de esquemas relacionales en proyecciones independientes, obteniéndose relaciones en FNBC (aunque en algunos casos se pueda producir pérdida de dependencias funcionales). Los algoritmos de síntesis aparecen como alternativa a los de descomposición, y "sintetizan" esquemas de relaciones en 3FN agrupando atributos afines.

La mayoría de los algoritmos de análisis y de síntesis exigen calcular previamente el recubrimiento irredundante y las claves del esquema de relación, los cuales, a su vez, se basan en el cálculo del cierre de un descriptor, por lo que todos estos algoritmos auxiliares se incluyen en la herramienta RENO y se describen en este capítulo.

Gran parte de los problemas de análisis que se abordan en este trabajo (determinar si un esquema de relación está en una determinada forma normal, descomposición de un esquema de relación, etc.), son de tipo NP-completo, lo cual se traduce en que la complejidad de los algoritmos que los implementan sea exponencial, por lo tanto es vital que los algoritmos base de los que parten (cierre de un descriptor, recubrimiento irredundante, etc.) sean lo más eficientes posible.

² DELPHI es una marca registrada.

³ RENO permite introducir un conjunto de atributos y de dependencias funcionales, devolviendo como resultado un conjunto de esquemas de relación en 3FN o en FNBC, las claves candidatas y, si se desea, los resultados intermedios (cierre de un descriptor, recubrimiento minimal, etc.); asimismo permite que el usuario introduzca los esquemas de relación que él mismo ha obtenido por aplicación de los procesos de síntesis o de análisis y comprueba si son o no correctos.

Como solución a estos problemas, DIEDERICH y MILTON (1988), en su artículo "New methods and fast algorithms for database normalization", propusieron nuevos algoritmos de normalización que consiguen disminuir los tiempos de computación para esquemas basados en modelos reales con un número elevado de dependencias, sin garantizar la misma disminución de tiempo para conjuntos de dependencias de laboratorio o experimentales. Presentamos aquí, además de los tradicionales, los nuevos algoritmos, junto con otras modificaciones propias que bien mejoran la eficiencia, bien la funcionalidad en otros casos.

Como algoritmo de normalización se ha elegido en primer lugar el de síntesis propuesto por BERNSTEIN (1976), que obliga a un cálculo previo del recubrimiento irredundante de un conjunto de dependencias, lo que, a su vez, hace preciso aplicar el cierre de un descriptor respecto a dicho conjunto. Aunque este algoritmo no exige el cálculo de claves (otros algoritmos de síntesis sí lo exigen, al igual que ocurre en los algoritmos de análisis), debido a la importancia que tienen las claves en el modelo relacional, hemos incluido también algoritmos para su determinación.

En el caso del algoritmo de análisis que normaliza esquemas en FNBC, el objetivo de la versión alternativa que proponemos es conseguir que en la descomposición se pierda el menor número de dependencias posibles, es decir, lograr que quede reflejada toda la semántica del esquema de relación original. En este caso, la versión alternativa puede ser menos eficiente que la tradicional, pero, desde el punto de vista del diseño, los resultados que proporciona este algoritmo son mejores.

A continuación aparece una versión tradicional de cada uno de los algoritmos (o la referencia donde se encuentra) y una versión "alternativa". Decimos "alternativa" y no "mejorada" porque en algunos casos, y debido al carácter NP-completo de los algoritmos, no ha sido posible encontrar ni en la bibliografía, ni tampoco analizando a fondo los algoritmos existentes, nuevas versiones que sean más eficientes. Lo que se ha pretendido conseguir en el caso de versiones alternativas que no mejoran la eficiencia es, bien mejorar la funcionalidad, bien demostrar que existen otros procedimientos diferentes de los ya existentes para llegar a los mismos resultados, aunque los tiempos de respuesta sean más o menos similares. Muchas de las versiones alternativas que se proponen se basan en un estudio previo de las dependencias del esquema de relación que se desea normalizar a fin de poder simplificar los cálculos posteriores.

Primeramente abordaremos el problema mediante la aproximación de síntesis, exponiendo los principales algoritmos tradicionales, y a continuación los nuevos algoritmos mejorados; también se hará un estudio comparativo analizando las razones por las cuales se consigue aumentar la eficiencia.

Después se abordará el proceso de análisis con una estructura distinta, ya que solamente se exponen los nuevos algoritmos.

2. ALGORITMOS RELATIVOS A LA NORMALIZACIÓN POR SÍNTESIS⁴

Como hemos visto en los dos capítulos precedentes, el objetivo final de todo proceso de normalización es obtener, a partir de un conjunto de atributos y de dependencias (en nuestro caso, sólo funcionales), un conjunto de esquemas normalizados.

El proceso que obtiene los esquemas normalizados tiene como requisito fundamental que el conjunto de dependencias de partida sea no redundante o, lo que es lo mismo, que sea un recubrimiento minimal. Para ello, es preciso la eliminación de atributos extraños y de dependencias redundantes, lo que se basa en el cálculo del cierre de un descriptor respecto de un conjunto de dependencias.

Comenzaremos, por tanto, viendo el algoritmo de cálculo del cierre de un descriptor, de cálculo del recubrimiento minimal, el de síntesis de Bernstein y expondremos un algoritmo de determinación de claves.

2.1. Algoritmo de cálculo del cierre de un descriptor

El cierre de un descriptor X respecto a un conjunto de dependencias DF , que representamos por X_{DF}^+ , tal como se ha definido en anteriores capítulos, es el conjunto de todos los atributos derivados de X usando las dependencias DF y aplicando el axioma de transitividad.

El algoritmo tradicional de cálculo del cierre de un descriptor fue propuesto por ULLMAN (1980) y se detalla en T1.

ALGORITMO T1 (Cierre de un descriptor)

Entrada:

Un conjunto de dependencias DF

Un descriptor X ⁵

⁴ Incluido el de determinación de claves.

⁵ Supondremos que los atributos de partida son todos los que aparecen en las dependencias funcionales, siendo X un subconjunto de los mismos. Si en X hubiese algún atributo que no apareciese en ninguna dependencia, el cálculo del cierre se realizaría de la misma manera y en él estaría incluido el (o los) atributo(s) que estuviese(n) en X aunque no se encontrasen en ninguna dependencia. En la actual versión de RENO, a diferencia de lo que ocurría en anteriores versiones, se introducen primero los atributos y, a continuación, las dependencias funcionales, de un modo sencillo y fácil de usar (*Lucas, friendb*) para el usuario.

Salida:

X^+ , cierre de X respecto a DF .

Proceso:

1) $X^+ = X$

2) Repetir hasta que no se añadan más atributos a X^+ .

Para cada dependencia $Y \rightarrow A$ en D

Si Y pertenece a X y A no está en X^+ entonces

$$X^+ = X^+ \cup A$$

La complejidad del algoritmo, en el peor de los casos, es $O(a.n^2)$, donde n es el número de dependencias de DF y a es el número de atributos.

El cierre de un descriptor es un cálculo básico y muy utilizado por el resto de los algoritmos, siendo su complejidad una de las causas del elevado coste computacional (y por tanto de la imposibilidad de aplicación práctica en bases de datos con un gran número de dependencias) de los algoritmos que lo utilizan. Hay dos métodos para mejorar el rendimiento de los algoritmos que utilizan el cierre de un descriptor:

- Mejorar el coste computacional del algoritmo mismo de cierre; por ejemplo, en BEERI (1979) se propone un algoritmo para el cálculo del cierre de un descriptor (que exponemos en el epígrafe 3.1.) en el cual el tiempo es lineal al ser accedida cada dependencia una sola vez.
- Evitar, siempre que sea posible, la computación de cierres en los algoritmos que los precisan (principalmente en el de recubrimiento minimal).

En DIEDERICH y MILTON (1988) se comprueba, mediante los correspondientes *bancos de pruebas* (*benchmarks*), que la ganancia obtenida aplicando el nuevo algoritmo de cierre no es suficiente para evitar los altos costes de computación del resto de los algoritmos, por lo que será necesario también reducir al máximo el número de cierres si se desea mejorar substancialmente la eficiencia de los algoritmos haciéndolos viables en la práctica.

2.2. Algoritmo de cálculo del recubrimiento minimal

El algoritmo de cálculo del recubrimiento minimal propuesto por ULLMAN (1980) y otros autores se detalla en T2.

ALGORITMO T2 (recubrimiento minimal)**Entrada:**

DF conjunto de dependencias elementales.

Salida:

H recubrimiento minimal de DF.

Proceso:

1) Eliminación de dependencias redundantes.

1.1) $H = D$

1.2) Repetir para cada dependencia $X \rightarrow A$ de DF:

$$G = H - \{ X \rightarrow A \}$$

Si A pertenece a X_G^+ entonces $H = G$

2) Eliminación de atributos extraños.

2.1) Repetir para cada dependencia $X \rightarrow B$ de H:

2.1.1) $L = X$

2.1.2) Repetir para cada atributo A de X

Si B pertenece a $(L - A)^+$

entonces $L = L - A$

2.1.3) Reemplazar $X \rightarrow B$ por $L \rightarrow B$

ATKINS (1988) demostró que el algoritmo, propuesto por ULLMAN (1980)⁶ y otros autores⁷ para el cálculo del recubrimiento minimal no es correcto, debido a que no siempre obtiene un recubrimiento minimal.

⁶ En ediciones posteriores de ULLMAN (1988) se corrigen los errores que se habían detectado en la definición de recubrimiento minimal y se llama la atención sobre el hecho de que los pasos de eliminación de atributos extraños y de dependencias redundantes tienen que ir en este orden y no pueden intercambiarse, pero no se incluye el algoritmo con las correspondientes correcciones.

⁷ En especial alguno que *depende funcionalmente* de Ullman.

Para demostrarlo basta con el siguiente ejemplo: Sea el conjunto de dependencias:

$$DF = \{ AB \rightarrow C, C \rightarrow B, A \rightarrow B \}$$

Si aplicamos el algoritmo expuesto anteriormente, comprobamos que ninguna dependencia es redundante, por lo tanto, en el paso 1 no se modifica el conjunto de dependencias. Al analizar los atributos extraños (paso 2 del algoritmo) comprobamos que el atributo B es extraño en la dependencia $AB \rightarrow C$ obteniéndose el siguiente conjunto como recubrimiento minimal:

$$\{ A \rightarrow C, C \rightarrow B, A \rightarrow B \}$$

que no cumple las condiciones necesarias para ser un conjunto minimal, ya que la dependencia $A \rightarrow B$ es redundante (se puede derivar de las dos primeras).

La corrección propuesta por Atkins consiste en intercambiar los dos pasos del algoritmo, es decir, primero eliminar los atributos extraños y después las dependencias redundantes, quedando el algoritmo T2 de la siguiente forma:

ALGORITMO T2 (recubrimiento minimal corregido)

Entrada:

DF conjunto de dependencias elementales.

Salida:

H recubrimiento minimal de DF.

Proceso:

1) Eliminación de atributos extraños.

1.1) Repetir para cada dependencia $X \rightarrow B$ de DF

1.1.1) $L = X$

1.1.2) Repetir para cada atributo A de X

Si B pertenece a $(L - A)^+$

entonces $L = L - A$

1.1.3) Reemplazar $X \rightarrow B$ por $L \rightarrow B$

2) Eliminación de dependencias redundantes.

$$2.1) H = F$$

2.2) Repetir para cada dependencia $X \rightarrow A$ de DF

$$G = H - \{ X \rightarrow A \}$$

Si A pertenece a X_G^+ entonces $H = G$

La corrección al algoritmo de Atkins, propuesta en NUMMENMAA y THANISCH (1990) no afecta a los algoritmos propuestos, ya que en ellos no se utiliza el cierre del conjunto de dependencias para analizar los atributos extraños.

2.3. Algoritmo de síntesis de Bernstein

BERNSTEIN (1976) propone, como alternativa al proceso de descomposición, un nuevo proceso por el cual, a partir de un conjunto de dependencias y de un conjunto de atributos, construye relaciones con la restricción de que todas las relaciones obtenidas estén en 3FN y de que el conjunto de esquemas de relación resultantes sea mínimo⁸.

El algoritmo propuesto se detalla a continuación.

ALGORITMO T3 (normalización por el método de síntesis)

Entrada:

DF conjunto de dependencias funcionales

Salida:

$\{A_i, DF_i\}$ conjunto de esquemas de relación en 3FN

Proceso:

- 1) Calcular el recubrimiento minimal H de DF
- 2) Realizar una partición de H en grupos, de tal forma que las dependencias de cada grupo tengan el mismo implicante.

⁸ Aunque el algoritmo de Bernstein se propone como objetivo diseñar un esquema relacional en el que el número de relaciones sea mínimo, sin embargo, en muchos casos no puede asegurarse que se cumpla este objetivo de minimalidad.

3) Unir claves equivalentes.

Hacer $J = \emptyset$

Para cada par de grupos H_i, H_j con partes izquierdas X_i, X_j respectivamente, si se cumple que:

X_i está incluido en X_j^+ y X_j está incluido en X_i^+

entonces : unir H_i y H_j ,

añadir a J las dependencias $X_j \rightarrow X_i$ y $X_i \rightarrow X_j$

borrar de H todas las dependencias que cumplan:

$X \rightarrow A$ si A pertenece a Y

$Y \rightarrow B$ si B pertenece a X

4) Eliminar dependencias transitivas

Encontrar un conjunto minimal $H' \subset H$ tal que:

$$(H' \cup J)^+ = (H \cup J)^+$$

Incluir cada dependencia de J a su correspondiente grupo de dependencias del conjunto H'

5) Construir relaciones

Para cada grupo construir un esquema de relación

$$R_i = \langle A_i, H_i \rangle$$

donde A_i es el conjunto de todos los atributos que aparecen en H_i .

Las manipulaciones que realiza el algoritmo son de naturaleza sintáctica y no tienen en cuenta la semántica; por ejemplo, dado el conjunto de dependencias:

$$R.A \rightarrow R.B$$

$$S.B \rightarrow S.C$$

$$T.A \rightarrow T.C$$

La tercera dependencia ($T.A \rightarrow T.C$) puede ser redundante o no, dependiendo del significado de las relaciones R, S y T; si aplicamos el algoritmo de Bernstein se considerará redundante, ya que no se reconocen los nombres de relación.

2.4. Algoritmo de determinación de claves

Aunque existen algunos algoritmos de determinación de claves —v.g. DELOBEL (1982)—, la mayoría de los autores se limitan a dar las definiciones de clave y de superclave sin proponer algoritmos para su cálculo; la razón es que, en general (si se exceptúan los ejemplos de laboratorio), en el mundo real suele ser bastante inmediato identificar las claves de una relación. En FERNANDEZ (1987) se propone un algoritmo para calcular todas las claves de un esquema; algoritmo bastante ineficiente puesto que repite innecesariamente cálculos (más adelante presentamos un nuevo algoritmo que mejora sustancialmente los tiempos de respuesta⁹).

Matriz de implicación

La matriz de implicación se utiliza para representar un conjunto de dependencias y se construye de la siguiente forma:

Dado un esquema $R(A,D)$, el conjunto D de dependencias funcionales formado por:

$$\{ X_i \rightarrow Y_i \}$$

en el que sin pérdida de generalidad supondremos:

$X_i \neq X_j$ para todo $i \neq j$ y que X_i y Y_i son disjuntos para todo i .

El conjunto DF puede representarse mediante una matriz M como la que se muestra en la figura 7.1.

Obsérvese la figura 7.1. donde $A_1, A_2, A_3, \dots, A_n$ (etiquetas de las columnas) son los atributos o elementos de A y $X_1, X_2, \dots, X_m$ (etiquetas de las filas) son los implicantes del conjunto DF. El elemento m_{ij} de M es:

$$\begin{cases} 1 & \text{si } A_i \text{ pertenece a } (X_i \cup Y_i) \\ 0 & \text{si } A_i \text{ no pertenece a } (X_i \cup Y_i) \end{cases}$$

⁹ En el capítulo 5 hemos presentado un procedimiento para determinar las claves de una relación que nos ha dado resultados correctos en todas las pruebas que hemos realizado. No hemos demostrado, sin embargo, que sea correcto, por lo que no estamos completamente seguros de que sea válido en todos los casos, por cuya razón no se ha incluido en este capítulo ni en RENO.

M		A1	A2	A3		An
	S1					
	S2					
	.					
	.					
	.					
	Sn					

Figura 7.1. Matriz de implicación

El cierre transitivo de M (M^+), se calcula de la siguiente forma:

- 1) $M^+ = M$
- 2) Repetir hasta que M^+ no cambie

Para todo $X_i \neq X_j$ de M^+

Si para todo A_k de X_j $m_{ik} = 1$ entonces

copiar todas las entradas a '1' de la fila X_i en los lugares homólogos de la fila X_j

Los "1" de la fila X_i del cierre de la matriz de implicación, M^+ , corresponden al cierre de X_i respecto de DF, de modo que el conjunto de dependencias que representa M^+ puede expresarse como:

$$M^+ = \{ X_i \rightarrow X_i^+ \}$$

Los "0" de la fila X_i del cierre de la matriz de implicación M^+ son los atributos Y_j no implicados por el descriptor X_i , de forma que se cumple:

$$(X_i \cup Y_j) \rightarrow A \quad \forall i$$

Por lo tanto, de acuerdo con la definición de clave que dimos en el capítulo 3, $(X_i \cup Y_j)$ forma una superclave del esquema; para que sea clave debe cumplir la propiedad de ser mínima.

En el algoritmo de cálculo de claves que vamos a exponer a continuación, la función *longitud de un descriptor* o número de atributos que tiene un descriptor X la representamos como $|X|$.

ALGORITMO T4 (cálculo de claves)

Entrada:

DF conjunto de dependencias funcionales

A conjunto de atributos

Salida:

K conjunto de claves del esquema

Proceso:

1) Calcular M^+ a partir del conjunto de dependencias DF.

2) $M1 = \emptyset$ y $M2 = \emptyset$

3) Construir $M1$,

$M1 = \{X_i \cup Y_i\}$ para $i = 1 \dots m$ donde

Y_i son los atributos con entrada a "0" de la fila i .

4) Si con $|Y_i| < 2$, $|Y_j| < 2$ se tiene que

$(X_i \cup Y_i) \subseteq (X_j \cup Y_j)$ para $i \neq j$, borrar la

entrada $(X_j \cup Y_j)$ de $M1$.

5) Si para las demás entradas de $M1$, $|Y_j| < 2$, el algoritmo concluye, $M1$ contiene todas las claves.

En caso contrario:

- 6) Para todo i , con $|Y_i| \geq 2$ calcular

$$a_{ij} = Y_i \setminus (X_j \cup Y_j) \text{ para todo } i \neq j \text{ y } |Y_j| \geq 0$$

- 7) Para todo i , borrar a_{il} si $a_{il} \supseteq a_{ij}$ con $j \neq l$

- 8) Introducir en M_2 las nuevas entradas $(X_i \cup a_{ij})$. Suprimir superconjuntos.

- 9) Para todo i en M_1 con $|Y_i| \geq 2$ y para todo j en M_2 con $X_i \neq X_j$ hallar:

$$a_{ijk} = Y_i \cap (X_j \cup a_{jk})$$

- 10) Para todo i , borrar a_{ilp} si $a_{ilp} \supseteq a_{ijk}$ con $(j,k) \neq (l,p)$.

- 11) Introducir en M_2 las nuevas entradas $(X_i \cup a_{ijk})$, si no son superconjuntos de descriptores ya en M_2 . Si se han creado nuevas entradas, ir a 9. Si no:

- 12) Copiar en M_2 los descriptores $(X_i \cup Y_i)$ de M_1 con $|Y_i| \leq 1$. Borrar superconjuntos.

M_2 contiene todas las claves.

Debido a la complejidad del algoritmo y a la cantidad de recursos de máquina que utiliza para esquemas con un gran número de dependencias, se opta por la división de los esquemas por medio de una partición funcional, de modo que se ahorre memoria y se tenga un tiempo de cálculo menor.

Partición funcional

El fundamento de la partición funcional es encontrar una relación entre dependencias funcionales que cumpla las propiedades de reflexividad, simetría y transitividad, de forma que produzca una partición de la relación original en clases de equivalencia.

Dado un esquema $R(A,DF)$, se obtiene una partición $R_i = (A_i, DF_i)$ con $i = 1 \dots n$, que cumple:

$$\begin{cases} A = A_1 \cup A_2 \cup \dots \cup A_n \\ DF = DF_1 \cup DF_2 \cup \dots \cup DF_n \end{cases}$$

K es una clave de $R(A,DF)$ si, y sólo si, se cumple:

$$K = (\cup K_i)_{i=1}^n$$

donde K_i es una clave del esquema $R_i = (A_i, DF_i)$.

El objetivo de la partición funcional es buscar una agrupación algorítmica de los atributos. Esto se podría haber logrado mucho más fácilmente si se hubiese partido de las entidades obtenidas de la agrupación semántica que proporciona el modelado conceptual; además, en este segundo caso, las entidades corresponden a objetos del mundo real, mientras que la agrupación algorítmica de los atributos puede carecer de todo significado, produciendo objetos totalmente ficticios.

3. NUEVOS ALGORITMOS DE NORMALIZACIÓN POR SÍNTESIS¹⁰

Los algoritmos que vamos a exponer a continuación están basados en los descritos en DIEDERICH y MILTON (1988), donde además se detalla su fundamentación matemática. El único algoritmo que no se encuentra en dicho artículo es el que proponemos para la determinación de las claves de un esquema¹¹.

Todos estos algoritmos tienen como objetivo fundamental reducir el tiempo computacional respecto a los propuestos anteriormente y, por tanto, hacerlos operativos. Lo que se pretende es primar esquemas de relación que representen modelos del mundo real, que son los que nos interesan principalmente, aunque para ello, no se mejoren o incluso se penalicen los ejemplos de laboratorio que no corresponden a situaciones reales.

Cuando diseñamos el esquema relacional que se ajusta a un determinado universo del discurso, las entidades y algunas interrelaciones se representan como relaciones. La característica principal que tienen las entidades es que uno o varios atributos forman el identificador o clave de dicha entidad y el resto de los atributos dependen de dicho identificador. Por ejemplo, las entidades EMPLEADO y DEPARTAMENTO las podemos representar en el modelo relacional como el siguiente esquema de relación:

EMPLEADO (*Cód_employado*, *Nom_employado*, *Dirección*, *Graduación*, *Salario*,
Cód_dept)

DEPARTAMENTO (*Cód_Dept*, *Nom_dept*, *Edificio*, *Planta*, *Área*)

Cod_employado → *Nom_employado*

¹⁰ Se incluye también el de determinación de claves.

¹¹ La versión mejorada del algoritmo que calcula las claves de un esquema de relación fue propuesto por A. de MIGUEL conjuntamente con PINILLA (1992), y supone una mejora muy importante en los tiempos de respuesta, incluso para esquemas de relación reales con un número elevado de dependencias; posteriormente en LÓPEZ (1998) hemos introducido mejoras en el mismo.

Cód_empleado → *Dirección*

Cód_empleado → *Graduación*

Cód_empleado → *Salario*

Cód_empleado → *Cód_dept*

Cód_dept → *Nom_dept*

Cód_dept → *Edificio*

Cód_dept → *Planta*

Cód_Dept → *Área*

Nom_dept → *Cód_dept*

Si observamos el esquema de relación del ejemplo, podemos ver que del conjunto de todos los atributos sólo un número muy reducido aparece en la parte izquierda de las dependencias, la mayoría de ellos se encuentran formando la parte derecha de las mismas. Si estudiamos la repercusión de esta característica en la aplicación de los algoritmos, podremos en muchos casos detectar de antemano dependencias redundantes y/o atributos extraños, descartándolos, y reduciendo así el número de dependencias y atributos a los que se aplica el algoritmo. Con este método los resultados pueden no ser satisfactorios para ejemplos de laboratorio, ya que se penalizan todos aquellos esquemas que no cumplan las condiciones antes expuestas, pero en situaciones reales el comportamiento de dichos algoritmos mejora considerablemente el consumo de recursos.

Como en todos los métodos algorítmicos, no se tiene en cuenta la semántica explícita del mundo real, sólo se considera la inherente a la representación matemática utilizada, por lo que los resultados serán tanto mejores cuanto más sencillos sean los esquemas de partida. Se puede considerar que el método de diseño más adecuado es el basado en el modelo E/R, porque produce ya en la transformación al modelo relacional esquemas más sencillos (menos atributos y dependencias) y con relaciones que representan elementos (entidades e interrelaciones) reales. Sin embargo, si se parte de la relación universal, se pierde semántica y los resultados que se obtienen, aunque matemáticamente sean correctos, pueden no ser los más adecuados para resolver el problema planteado; por ejemplo en CODD (1990) se muestran algunos de los problemas asociados a la relación universal.

3.1. Algoritmo de cálculo del cierre de un descriptor

Comenzamos viendo el cálculo del cierre de un descriptor respecto a un conjunto de dependencias, por ser éste un cálculo básico y del que los algoritmos siguientes van a hacer uso.

Como ya mencionamos, el problema principal del algoritmo propuesto por ULLMAN (1980) (que se ha expuesto en T1) es su coste y su complejidad cuadrática. La causa fundamental de estos problemas es que a partir de $X^+ = X$ se recorre varias veces (tantas como pasos tenga el algoritmo) el conjunto de dependencias, añadiendo la parte derecha de la dependencia siempre que la parte izquierda esté incluida en X^+ .

El algoritmo que a continuación detallamos, DIEDERICH (1988), a diferencia del anterior, se caracteriza por tener una complejidad lineal de $O(n)$, siendo n el número de dependencias.

ALGORITMO N1 (nuevo cierre de un descriptor)

Entrada:

DF conjunto de dependencias funcionales.

X conjunto de atributos

Salida:

X^+ cierre de X respecto a DF

Proceso:

1) $X^+ = X$, ACTUAL = X

2) Para cada dependencia funcional d de DF

contador (d) = N° de atributos de la parte izquierda de d .

3) Para cada atributo A, construir

lista (A) = $\{(Y \rightarrow B) \in DF / A \in Y\}$ ¹²

¹² Lista (A) es, por tanto, una lista que contiene las dependencias funcionales en las que A es implicante (en lugar de las dependencias podría contener punteros a las mismas).

- 4) Mientras $ACTUAL \neq \emptyset$
- 5) Seleccionar y eliminar un atributo A de ACTUAL
- 6) Para cada dependencia $d = Y \rightarrow C$ de lista(A)
- 7) Decrementar el contador (d)
- 8) Si contador(d) = 0 entonces

Si C no está en X^+

$X^+ = X^+ \cup \{C\}$

$ACTUAL = ACTUAL \cup \{C\}$

La idea principal en la que se basa el algoritmo es añadir la parte derecha de las dependencias al cierre sólo cuando se tiene la certeza de que todos los atributos que forman la parte izquierda pertenecen a dicho cierre. De esta forma, cada dependencia es accedida una sola vez, por lo que la complejidad del algoritmo es lineal $O(n)$, siendo n el número de dependencias.

Puede parecer que introducir la estructura lista(A) agrava el coste computacional del cálculo del cierre, pero este coste adicional se va a ver amortizado al utilizarse por el resto de los algoritmos que intervienen en el cálculo del recubrimiento minimal. No debemos olvidar que la obtención del cierre transitivo no es sino un instrumento del que se sirven el resto de los algoritmos, si bien también es esta una de las causas del elevado coste computacional de dichos algoritmos.

Aunque la utilización del nuevo algoritmo de cálculo del cierre de un descriptor introduce mejoras en los tiempos del resto de los algoritmos, como ya hemos advertido, éstas no son suficientes para hacerlos operativos.

A continuación vamos a ver qué otras mejoras se pueden incluir en los algoritmos del cálculo del recubrimiento minimal, tal como ha sido propuesto en DIEDERICH (1988).

El cálculo de un conjunto de dependencias minimal, o no redundante, se basa en la eliminación de atributos extraños y de dependencias redundantes. En esta sección vamos a ver ambos algoritmos por separado para poder estudiar con mayor detenimiento cada uno de ellos.

Antes de comenzar a exponer los algoritmos es necesario definir los conceptos en los que se basan.

3.2. Algoritmo de cálculo del recubrimiento minimal

A) Nuevos conceptos

Definiciones:

$$\text{DepForLHS}^{13} (X) = \{X \rightarrow A \in DF\}$$

$\text{DepForLHS} (X)$ no contiene dependencias triviales ni duplicadas

$$\text{LHS} (DX) = \{X / \text{DepForLHS}(X) \neq \emptyset\}$$

ro-atributo¹⁴: A es ro-atributo si A solo aparece en la parte derecha de las dependencias.

rl-atributo: B es rl-atributo si no es ro-atributo.

$$\text{RO}_X = \{A / X \rightarrow A \text{ pertenece a DF y A es ro-atributo}\}$$

$$\text{RL}_X = \{A / X \rightarrow A \text{ pertenece a DF y A es rl-atributo}\}$$

$$\text{R}_X = \text{RO}_X \cup \text{RL}_X$$

El **r-cierre** de X respecto a un conjunto de dependencias G, lo representamos como X_G^+ , es el conjunto de todos los atributos A tales que $Y \rightarrow A$ pertenece a G e Y pertenece a X_G^+ . Por lo tanto, X_G^+ es el conjunto de todos los atributos de las partes derechas de las dependencias derivados de X respecto a G. Es fácil ver que, para un conjunto de atributos X, X_G^+ está incluido en X_G^+ y los únicos atributos que pueden no encontrarse en X_G^+ son los que forman X.

El r-cierre aporta más información que el cierre transitivo de un descriptor y es especialmente útil para la eliminación de atributos extraños implicados y dependencias redundantes.

B) Atributos extraños

Los atributos extraños se pueden dividir en dos tipos:

- Implicados.

B es un atributo extraño implicado en la dependencia $X \rightarrow A$, si B es atributo extraño y además cumple que: si $X = ZB$, $X \neq B$, entonces B pertenece a Z_{DF}^+ .

¹³ LHS: son las siglas de *Left Hand Side* (parte izquierda).

¹⁴ Ro: son siglas de *Right Only* (sólo a la derecha).

- No implicados

B es un atributo extraño no implicado en la dependencia $X \rightarrow A$, si B es atributo extraño y además cumple que: si $X = ZB$, $X \neq Z$, entonces B no pertenece a Z^+_{DF} .

La diferencia estriba en que en una dependencia que contenga un atributo extraño no implicado se puede afirmar que, después de la eliminación del atributo extraño, la dependencia resultante es una dependencia redundante. Por ejemplo, dada la dependencia $X \rightarrow A$ de DF, donde $X = ZB$ y $X \neq Z$, si B es un atributo extraño en dicha dependencia se cumple que A pertenece a Z^+_{DF} , si además B es un atributo extraño no implicado, según la definición anterior tenemos que B no pertenece a Z^+_{DF} , por lo tanto tenemos una nueva dependencia $Z \rightarrow A$ y podemos afirmar que A pertenece a Z^+_{DF} , lo que indica que la dependencia resultante después de eliminar el atributo extraño es una dependencia redundante en el conjunto DF.

C) Eliminación de atributos extraños

Como ya hemos mencionado, para que el cálculo del recubrimiento minimal sea correcto, primero se han de eliminar los atributos extraños y, como segundo paso, se han de eliminar las dependencias redundantes.

En estos algoritmos el orden se modifica algunas veces, ya que es posible eliminar atributos extraños no implicados al mismo tiempo que dependencias redundantes, según explicamos anteriormente, reduciendo de este modo el número de atributos que deben ser comprobados como posibles atributos extraños. En cambio, los atributos extraños implicados deben ser eliminados antes que las dependencias redundantes para que el cálculo sea correcto. Vamos a ver la afirmación anterior con un ejemplo propuesto en DIEDERICH (1988):

Dado el conjunto de dependencias:

$BAC \rightarrow D$

$E \rightarrow A$

$BAC \rightarrow E$

$BAC \rightarrow G$

$BAC \rightarrow F$

$EH \rightarrow A$

$DH \rightarrow I$

Si aplicamos el algoritmo tradicional es necesario calcular 8 cierres para detectar el atributo A como extraño en el conjunto $\{A,B,C\}$ y eliminarlo. Según el algoritmo que se propone es posible detectar que A es el único atributo candidato a ser extraño en el conjunto $X = \{A,B,C\}$ y sólo se calculará un cierre para las cuatro dependencias $\{BAC \rightarrow D, BAC \rightarrow E, BAC \rightarrow G, BAC \rightarrow F\}$. Además, el atributo extraño no implicado H de las dependencias $EH \rightarrow A$ será eliminado a la vez que se eliminan las dependencias redundantes.

Cuando definimos el r-cierre, mencionamos que iba a ser especialmente útil en la eliminación de atributos extraños y dependencias redundantes. Como en este momento el tema que estamos tratando son los atributos extraños, vamos a ver dos proposiciones que relacionan a ambos.

- (1) Si C pertenece a X y C es un atributo extraño implicado en $X \rightarrow A$, entonces C pertenece a $(X \cup RL_X)^+_G$
- (2) Si C pertenece a X y C no pertenece a $(X \cup RL_X)^+_G$ entonces C no es un atributo extraño en $X \rightarrow A$ ó $X \rightarrow A$ es redundante.

La idea principal del algoritmo es eliminar todos los atributos extraños implicados para cada conjunto de dependencias con igual parte izquierda, de esta forma no se tratan dependencias individualmente. Además, sólo los atributos que cumplan la propiedad (1) serán comprobados como posibles atributos extraños.

ALGORITMO N2: (eliminación de atributos extraños)

Entrada:

DF, conjunto de dependencias funcionales elementales.

Salida:

H, conjunto de dependencias sin atributos extraños implicados.

Proceso:

Para cada $DepForLHS(X)$ Si $|X| > 1$

1) Calcular $rc(X) = (X \cup RL_X)^+_{DF}$

2) Mientras $|X| > 1$ por cada B de X

Si B pertenece a $X \cap rc(X)$ entonces

$X' = X - B$

calcular X'^{+}_{DF}

Si B pertenece a X'^{+}_{DF} entonces $X = X'$

3) Si X ha cambiado en el paso 2

Reemplazar X por el nuevo valor X' en todas las dependencias de $\text{DepForLHS}(X)$.

Podemos ver en el algoritmo que es necesario calcular el r-cierre, (que tiene complejidad similar al cierre), para cada uno de los grupos de dependencias y, además, por cada dependencia transitiva, se calcula un cierre adicional. Por lo tanto, en el peor de los casos, el tiempo invertido por este algoritmo para eliminar los atributos extraños puede llegar a ser considerablemente mayor que el que invierte el algoritmo tradicional. Pero, como ya hemos mencionado, el objetivo principal que se persigue es hacer viable la aplicación de dichos algoritmos a esquemas reales, los cuales se caracterizan por tener un elevado número de dependencias en las que suelen coincidir sus implicantes. Si aplicamos el algoritmo antes expuesto a un esquema de estas características, se puede ver que el número de cierres y r-cierres calculados es considerablemente menor que los que se calcularían el algoritmo tradicional y, por consiguiente, el tiempo invertido también será inferior.

D) Eliminación de dependencias redundantes

Es el siguiente paso a realizar para el cálculo del recubrimiento minimal.

El mayor problema del algoritmo de eliminación de dependencias redundantes tradicional es que los mismos cálculos se repiten varias veces, por ejemplo:

Dado el conjunto de dependencias (propuesto en DIEDERICH, 1988):

$AB \rightarrow C$

$AB \rightarrow D$

$AB \rightarrow E$

$AB \rightarrow F$

$C \rightarrow E$

$C \rightarrow J$

Para la eliminación de dependencias redundantes de este ejemplo, el cálculo del cierre de AB se repite varias veces, cada vez que se trata una dependencia cuyo implicante es AB. Parece obvio que si agrupamos las dependencias con igual parte izquierda y hacemos los cálculos por grupos de dependencias mejoraríamos el tiempo.

Por otro lado, observando las características de las dependencias, algunas de ellas pueden ser descartadas y no pasar la prueba de dependencias redundantes. En el ejemplo anterior, la segunda, cuarta y sexta dependencia nunca pueden ser redundantes, ya que sus partes derechas ni están incluidas en ninguna parte izquierda de las otras dependencias ni tampoco en su cierre; por lo tanto deben eliminarse del conjunto de dependencias que vayan a ser comprobadas como dependencias redundantes.

Al igual que en la eliminación de atributos extraños, el r-cierre también puede ser utilizado para eliminar dependencias redundantes por las siguientes proposiciones:

- (1) $X \rightarrow A$ es redundante si

$$A \in (X \cup RL_X)^+_G \cap RO_X$$

- (2) $X \rightarrow A$ no es redundante si

$$A \notin (X \cup RL_X)^+_G$$

Mediante la proposición (1) reducimos el número de dependencias que tienen que ser comprobadas como redundantes y, en consecuencia, se reduce el número de veces que se debe aplicar el algoritmo tradicional. Al ser la proposición (1) una condición necesaria y no suficiente, no se puede afirmar que las dependencias que no cumplan dicha condición no sean redundantes y será necesario aplicar el método tradicional para comprobarlo.

A continuación detallamos el algoritmo propuesto para la eliminación de dependencias redundantes.

ALGORITMO N3 (eliminación de dependencias redundantes)

Entrada:

DF conjunto de dependencias sin atributos extraños implicados.

Salida:

H recubrimiento minimal de DF

Proceso:

$$0) \quad H = DF$$

$$1) \quad \text{Para cada DepForLHS}(X)$$

$$2) \quad G = \{H - \text{DepForLHS}(X)\}$$

$$\text{crc}(X) = \emptyset.$$

$$\text{Attr} = X.$$

$$3) \quad \text{Para cada dependencia } X \rightarrow A \text{ de DepForLHS}(X)$$

Si A pertenece a $\text{crc}(X)$,

borrar $X \rightarrow A$ de H

si no

Si A es un rl-atributo

$$\text{Attr} = \text{Attr} \cup \{A\}$$

$$\text{crc}(X) = \text{Attr}_G^{\wedge}$$

Si $\text{crc}(X) = \emptyset$

$$\text{crc}(X) = X_G^{\wedge}$$

$$4) \quad \text{Para cada dependencia } X \rightarrow A \text{ de}$$

$$\text{DepForLHS}(X) \cap H$$

$$5) \quad \text{Si A pertenece a } \text{crc}(X)$$

Si A es ro-atributo

ó

si A es rl-atributo y A pertenece X^+ respecto $H - \{X \rightarrow A\}$

Borrar $X \rightarrow A$ de H

Si observamos el algoritmo, en el paso 3 se eliminan las dependencias redundantes que cumplen la proposición (1) y además se calcula el r-cierre de todos los atributos implicados por X que aparecen en la parte izquierda de las dependencias, en el algoritmo se representa por $\text{crc}(X)$. De esta forma, se va preparando el siguiente paso.

En el paso 4 sólo se comprueba si son dependencias redundantes aquellas cuyo implicado pertenezca al cierre calculado anteriormente, $\text{crc}(X)$; si la dependencia no pertenece a dicho cierre, quiere decir que en ningún caso esa dependencia puede ser redundante (proposición (2)).

Las dependencias que en el paso 4 pueden ser redundantes, sólo serán comprobadas cuando el atributo implicado forme parte de algún implicante del conjunto de dependencias (rl-atributo), y para esto se aplica el método tradicional. En el caso contrario, es decir, el atributo sólo forma parte de los atributos implicados (ro-atributo), se puede afirmar directamente que la dependencia es redundante.

E) Algoritmo completo de cálculo del recubrimiento minimal

Una vez vistos los algoritmos de eliminación de atributos extraños y eliminación de dependencias redundantes, estamos en condiciones de exponer el algoritmo de cálculo del recubrimiento minimal, el cual se basa en los dos algoritmos anteriores.

ALGORITMO: N4 (nuevo recubrimiento minimal)

Entrada:

DF conjunto de dependencias elementales.

Salida:

H recubrimiento minimal de DF.

Proceso:

- 1) Algoritmo de eliminación de atributos extraños implicados (Algoritmo N2).
- 2) Reconstruir los conjuntos $\text{DepForLHS}(X)$, eliminando dependencias duplicadas.
- 3) Eliminar dependencias redundantes (Algoritmo N3).

Vamos a ver el funcionamiento de este algoritmo con un ejemplo:

Dado el conjunto de dependencias:

$$ABC \rightarrow K \quad AB \rightarrow D$$

$$BN \rightarrow H \quad B \rightarrow A$$

$$AB \rightarrow E \quad ABC \rightarrow D$$

$$B \rightarrow H \quad AB \rightarrow G$$

$$AB \rightarrow F \quad ABC \rightarrow J$$

$$B \rightarrow G \quad D \rightarrow A$$

Comenzamos aplicando el algoritmo de eliminación de atributos extraños (ALGORITMO N2).

- Para DepForLHS(AB)

Paso 1: $rc(AB) = \{ A, B, D \} \wedge G = \{ A, H, G \}$.

Paso 2: Sólo A puede ser un atributo extraño y A pertenece a B^+_{DF} , por lo tanto es eliminado.

Paso 3: AB es reemplazado por B en todas las dependencias de DepForLHS(AB).

- Para DepForLHS(ABC)

Paso 1: $rc(ABC) = \{ A, B, C, D \} \wedge G = \{ D, E, F, G, A, H \}$

Paso 2: Sólo A puede ser un atributo extraño y A pertenece a $BC+DF$, por lo tanto es eliminado.

Paso 3: ABC es reemplazado por BC en todas las dependencias de DepForLHS(ABC).

- Para DepForLHS(BN)

Paso 1: $rc(BN) = \{ B, N \} \wedge G = \{ D, E, F, G, A, H \}$

Paso 2: Ningún atributo puede ser atributo extraño implicado (N es un atributo extraño no implicado)

Paso 3: No se produce ningún cambio.

Por lo tanto, el conjunto de dependencias resultantes, después de eliminar los atributos extraños implicados, es:

$$\begin{array}{lll}
 B \rightarrow D & B \rightarrow F & BC \rightarrow D \\
 B \rightarrow A & B \rightarrow G & BC \rightarrow J \\
 B \rightarrow E & D \rightarrow A & BC \rightarrow K \\
 B \rightarrow H & & BN \rightarrow H
 \end{array}$$

Pasamos a aplicar el algoritmo de eliminación de dependencias redundantes (ALGORITMO N3).

- Para DepForLHS(B)

Paso 2: $G = \{ BC \rightarrow D, BC \rightarrow J, BC \rightarrow K,$

$$D \rightarrow A, BN \rightarrow H \}$$

$$\text{Attr} = \{ B \}$$

$$\text{crc}(B) = \emptyset$$

Paso 3: Sólo la dependencia $B \rightarrow A$ es eliminada y $\text{crc}(B) = [B, D, A]$

Paso 4: Sólo la dependencia $B \rightarrow D$ se comprueba si es redundante, siendo necesario calcular el cierre de B respecto a $H - \{B \rightarrow D\}$, lo que da como resultado que no es redundante.

- Para DepForLHS(BC)

Paso 2: $G = \{ B \rightarrow D, B \rightarrow E, B \rightarrow F, B \rightarrow G, B \rightarrow H,$

$$D \rightarrow A, BN \rightarrow H \}$$

$$\text{Attr} = \{ B, C \}$$

$$\text{crc}(B C) = \emptyset$$

Paso 3: Ninguna dependencia es eliminada y $\text{crc}(B C) = [D, E, F, G, H, A]$

Paso 4: Sólo la dependencia $B C \rightarrow D$ es comprobada por si es redundante, para lo que es necesario calcular el cierre de BC respecto a $H - \{B C \rightarrow D\}$, dando como resultado que es redundante y, por lo tanto, se elimina.

- Para DepForLHS(BN)

Paso 2: $G = \{ B \rightarrow D, B \rightarrow E, B \rightarrow F, B \rightarrow G, B \rightarrow H, \\ D \rightarrow A, BC \rightarrow J, BC \rightarrow K \}$

Attr = $\{ B, N \}$

crc (B C) = $\emptyset$

Paso 3: Ninguna dependencia es eliminada y crc (B N) = $[D, E, F, G, H, A]$

Paso 4: la dependencia $B N \rightarrow H$ se comprobará si es redundante y se elimina sin necesidad de calcular el cierre de BN.

- Para DepForLHS(D)

Paso 2 : $G = \{ B \rightarrow D, B \rightarrow E, B \rightarrow F, B \rightarrow G, B \rightarrow H, \\ BC \rightarrow J, BC \rightarrow K \}$

Attr = $\{ D \}$

crc (D) = $\emptyset$

Paso 3: Ninguna dependencia es eliminada y crc (D) = $\emptyset$

Paso 4: No es necesario tratar este paso, ya que crc (D) es $\emptyset$.

El conjunto de dependencias resultantes, después de eliminar las dependencias redundantes, es decir, el recubrimiento minimal, es

$B \rightarrow D$	$B \rightarrow F$	$BC \rightarrow J$
$B \rightarrow E$	$B \rightarrow G$	$BC \rightarrow K$
$B \rightarrow H$	$D \rightarrow A$	

Sólo ha sido necesario calcular 5 cierres y 6 r-cierres frente a los 19 que habríamos necesitado si aplicamos el algoritmo tradicional, lo cual supone un ahorro considerable de tiempo y memoria que aumenta aún más en conjuntos con un número elevado de dependencias.

3.3. Algoritmo de síntesis

Una causa de la lentitud del algoritmo de síntesis, propuesto por Bernstein, es la búsqueda de descriptores equivalentes entre los distintos grupos de dependencias. Para

detectar si un implicante tiene o no un implicante equivalente en el conjunto de dependencias, es necesario calcular el cierre de todos los implicantes de dicho conjunto.

DIEDERICH y MILTON (1988) demuestran dos teoremas en los que se basan para proponer un algoritmo de síntesis que trata de manera más eficiente la obtención de descriptores equivalentes y la determinación de dependencias redundantes. A continuación presentamos dicho algoritmo.

ALGORITMO N5 (nueva normalización por síntesis)

Entrada:

DF conjunto de dependencias funcionales.

Salida:

$R(A_i, DF_i)$, conjunto de esquemas en 3FN.

Proceso:

- 1) Calcular el recubrimiento minimal H de DF, aplicando el algoritmo N4, y guardar los $crc(X)$ para todo X.
- 2) Agrupar dependencias con igual parte izquierda (construir $DepForLHS(X)$, para todo X).
- 3) Reagrupar dependencias.

$$J = \emptyset.$$

Por cada implicante X que cumpla:

$$X \cap (X \cup RL_X)^+_G \neq \emptyset$$

determinar si X tiene una parte izquierda equivalente en H, es decir, si existe un Y que cumpla:

$$X \subset Y^+_{DF}, Y \subset X^+_{DF}.$$

Si se encuentra un implicante equivalente Y

- Agrupar los grupos de dependencias de X e Y en uno solo.
- Añadir las dependencias $X \rightarrow Y, Y \rightarrow X$ a J.

- Borrar de H las dependencias

$X \rightarrow A$ si A pertenece a Y

$Y \rightarrow B$ si B pertenece a X

4) Eliminar dependencias redundantes.

Por cada dependencia $X \rightarrow A$ de H.

Si A es un rl-atributo y cumple que A pertenece a $(X \cup RL_X)^+_G$

(o lo que es lo mismo A pertenece a $crc(X)$).

Comprobar si $X \rightarrow A$ es redundante en $H \cup J$, es decir, calcular el cierre de X respecto a $(H \cup J) - \{X \rightarrow A\}$.

Si A pertenece a X^+

Eliminar la dependencia $X \rightarrow A$ de H.

Añadir cada dependencia de J a su grupo correspondiente.

5) Formar relaciones.

Cada grupo de dependencias forma una relación en 3FN, donde el conjunto de atributos de cada relación es el resultado de la unión de los atributos que forman las dependencias.

En este caso, al igual que en los algoritmos anteriores, lo que se intenta es, en los pasos más costosos del algoritmo original, ver qué condiciones se han de cumplir para poder saltarlos. Es por esto que en grupos reducidos de dependencias no se pueden apreciar las mejoras, al ser igual de costoso evaluar una condición que realizar el cálculo; por el contrario, en conjuntos con un número elevado de dependencias es donde mejor se pueden apreciar las mejoras.

3.4. Algoritmo de determinación de claves

Una de las objeciones que se puede hacer al algoritmo de cálculo de claves presentado anteriormente es la repetición sucesiva de cálculos que es necesario realizar y el coste en tiempo y memoria que ello supone. Ésta es la razón que nos ha llevado

—véase PINILLA (1992)— a buscar una forma más eficiente para la determinación de las claves de un esquema que salvase los inconvenientes del algoritmo anterior.

La primera mejora que se introduce proviene de la observación de determinadas características que cumplen los atributos que forman parte de una clave. Dicha característica la podemos enunciar de la siguiente forma:

"Si en el conjunto de atributos del esquema alguno de ellos aparece únicamente como implicante (o parte izquierda) del conjunto de dependencias o no aparece en ninguna de las dependencias del esquema, dicho atributo o atributos forman parte de todas las claves del esquema".

La afirmación anterior parece obvia, ya que si existe algún atributo que interviene en la determinación de otros atributos y no es determinado por ningún otro, para que se cumpla que toda clave del esquema determine mínimamente al conjunto de atributos, es necesario que dicho atributo forme parte de todas las claves. Lo mismo ocurre cuando dicho atributo no es ni implicante ni implicado, aunque éste es un caso que en la realidad no suele producirse.

Comprobaremos dicha afirmación con el siguiente ejemplo:

Dado el esquema $R(AT, DF)$ donde:

$$AT = \{ A, B, C, D, E \}$$
$$DF = \{ AB \rightarrow C, B \rightarrow D, D \rightarrow E, E \rightarrow B \}$$

Podemos observar que el atributo A sólo aparece en la parte izquierda de las dependencias del esquema, luego ha de formar parte de todas las claves.

Aquellos atributos que van a formar parte de todas las claves pueden ser eliminados de los implicantes de las dependencias a fin de simplificar los cálculos.

La proposición anterior no resuelve todos los casos, sólo reduce el problema. La reducción puede ser considerable si el esquema es el resultado del modelado de un problema real, ya que en él un solo atributo o dos determinan, en general, el resto de los atributos y no suelen venir implicados por ningún otro.

Al comienzo de esta sección decíamos que el problema del algoritmo propuesto en el apartado 2.4 era la sucesiva repetición de cálculos que realiza. Si estudiamos detenidamente dicho algoritmo, se puede observar que el conjunto M1, en el paso 3, está formado por conjuntos de atributos que son superclaves. Para ser claves, les falta cumplir la condición de ser mínimos. Por lo tanto, el objetivo del resto de los pasos es encontrar los conjuntos de atributos mínimos que sigan cumpliendo la propiedad de ser

claves; para ello se apoya en un segundo conjunto, M2, donde se introducen subconjuntos de los anteriores. El algoritmo finaliza cuando ya no es posible obtener más subconjuntos distintos.

El algoritmo que proponemos a continuación se basa en la misma idea, es decir, a partir de un conjunto que contiene superclaves, encontrar subconjuntos que sigan cumpliendo la propiedad de determinar al resto de los atributos, además de ser mínimos, es decir, que sean claves; pero, a diferencia del anterior, se eliminan pasos y operaciones innecesarios.

ALGORITMO N6 (nuevo algoritmo de determinación de claves)

Entrada:

$R(A, DF)$ Esquema de relación

Salida: .

$\{K_i\}_n$, conjunto de las claves del esquema

Proceso:

1) Calcular M^+ , a partir del conjunto de dependencias DF.

2) $M1 = \emptyset$

3) Construir M1,

$M1 = \{ X_i \cup Y_i \}$ para $i = 1 \dots m$ donde

Y_i son los atributos con entrada a '0' de la fila i .

4) Si con $|Y_i| < 2$, $|Y_j| < 2$ se tiene que

$(X_i \cup Y_i) \subseteq (X_j \cup Y_j)$ para $i \neq j$, borrar la entrada $(X_j \cup Y_j)$ de M1.

5) Si para las demás entradas de M1, $|Y_j| < 2$, el algoritmo concluye, **M1 contiene todas las claves:**

En caso contrario:

6) Para todo i , con $|Y_i| \geq 2$ calcular

$a_{ij} = Y_i \cap (X_j \cup Y_j)$ para todo $i \neq j$ y $|Y_j| \geq 0$

- 7) Para todo i , borrar a_{ij} si $a_{ii} \geq a_{ij}$ con $j \neq i$
- 8) Sustituir las entradas $(X_i \cup Y_i)$ con $|Y_i| > 2$ por los nuevos $(X_i \cup a_{ij})$
- 9) Si todas las entradas en $M1$ son de la forma $(X_i \cup Y_i)$ con $|Y_i| \leq 1$, o no se han introducido nuevas entradas en el paso 8: Suprimir superconjuntos, y el **conjunto $M1$ contiene todas las claves**. En caso contrario volver a 6.

Se han reducido en 3 los pasos respecto al algoritmo original, pero lo más importante es que no se dan vueltas innecesarias, y que la repetición de cálculos es mínima. Por ejemplo, en el algoritmo original las operaciones que se realizan en el paso 6 y 9 son similares y, además, en el paso 9 se repiten todas las del paso 6 más las suyas propias.

Otro inconveniente del algoritmo original, que salva el nuevo algoritmo, es que si en el paso 6 —del algoritmo original— ya se han encontrado todas las claves, es necesario continuar y realizar el resto de los pasos hasta concluir, mientras que en el algoritmo que proponemos, como en cada pasada se realiza una única minimización, en el momento en que se encuentren todas las claves concluye.

Para este algoritmo sigue siendo válida la realización de una partición funcional.

Con el objeto de poder comprender más claramente el funcionamiento del algoritmo, vamos a aplicarlo al siguiente ejemplo:

Sea el esquema de relación:

$R(A, B, C, D, E, G)$

$AB \rightarrow C$	$C \rightarrow A$
$BC \rightarrow D$	$D \rightarrow E$
$BE \rightarrow C$	$D \rightarrow G$
$CG \rightarrow B$	$CE \rightarrow G$

La matriz de implicación correspondiente es:

M:	A	B	C	D	E	G
AB	1	1	1	0	0	0
BC	0	1	1	1	0	0
BE	0	1	1	0	1	0
CG	0	1	1	0	0	1
CE	0	0	1	0	1	1
C	1	0	1	0	0	0
D	0	0	0	1	1	1

Paso 1: Cálculo de M^+

M:	A	B	C	D	E	G
AB	1	1	1	0	0	0
BC	0	1	1	1	0	0
BE	0	1	1	0	1	0
CG	0	1	1	0	0	1
CE	0	0	1	0	1	1
C	1	0	1	0	0	0
D	0	0	0	1	1	1

Paso 3: $M_1 = [AB, BC, BE, CG, CE, C \cup BDEG, D \cup ABC]$

Paso 6: $a_{61} = B$ $a_{62} = B$ $a_{63} = BE$ $a_{64} = G$ $a_{65} = E$ $a_{67} = BD$
 $a_{71} = AB$ $a_{72} = BC$ $a_{73} = B$ $a_{74} = C$ $a_{75} = C$ $a_{76} = CB$

Paso 7: Se eliminan por ser superconjuntos a_{62} , a_{63} , a_{67} , a_{71} , a_{72} , a_{75} , a_{76}

Paso 8: El nuevo conjunto M_1 queda:

$M_1 = [AB, BC, BE, CG, CE, CB, CG, CE, DB, DC]$

Paso 9: Eliminamos superconjuntos y el algoritmo concluye con:

$M_1 = [AB, BC, BE, CG, CE, DB, DC]$

Para terminar vamos a exponer cómo quedaría el algoritmo de determinación de claves completo.

ALGORITMO N7 (nuevo algoritmo completo de determinación de claves)

Entrada:

$R(A,D)$ Esquema de relación

Salida:

$\{K_i\}_n$ conjunto de las claves del esquema

Proceso:

0) $K = \emptyset$.
 $D' = DF$

- 1) Para cada uno de los atributos A_i de A .

Si A_i no aparece en ninguna dependencia de D'

o

si sólo aparece en la parte izquierda de las dependencias de D' .

$$K = K \cup \{ A_i \}$$

Eliminar A_i de todas las dependencias de D' en las que aparezca, si en alguna era el único implicante, eliminar dicha dependencia de D' .

- 2) Realizar una partición funcional sobre D'
- 3) Para cada partición $\{A_i, DF_i\}$ obtenida en 2
 - 3.1) Aplicar el algoritmo N6
 - 3.2) Las claves de la partición $\{A_i, DF_i\}$ serán $\{K_{ij}\}_{j=1}^n$
- 4) Formar las claves del esquema como la concatenación de las claves de las particiones y el conjunto K obtenido en el paso 1.

4. ALGORITMOS QUE DETERMINAN LA FORMA NORMAL EN LA QUE SE ENCUENTRA UN ESQUEMA DE RELACIÓN

Antes de proceder a la presentación de los algoritmos de análisis, vamos a exponer algunos algoritmos que permiten determinar la forma normal en la que se encuentra un esquema de relación. Se trata de tres algoritmos, el primero de los cuales comprueba si el esquema está en 2FN, el segundo si está en 3FN y el último verifica si está en FNBC¹⁵.

4.1. Determinación de 2FN

Definición:

Un esquema de relación $R(AT, DEP)$ está en Segunda Forma Normal (2FN) si, además de estar en 1FN, cada atributo no principal tiene dependencia funcional completa respecto de cada una de las claves de la relación.

¹⁵ En RENO, además de estos algoritmos, se encuentran implementados los propuestos en FERNÁNDEZ, (1987). En LÓPEZ (1998) se comparan los distintos algoritmos y se analizan los tiempos de proceso de unos y otros, llegando a la conclusión de la mayor eficiencia de los que aquí presentamos.

Por la propia definición de 2FN, un esquema de relación **no se encuentra en 2FN**, en cuanto que exista algún subconjunto propio de la(s) clave(s) del esquema en cuyo cierre transitivo aparezcan atributos no principales.

Basándonos en esta idea, el algoritmo que seguidamente proponemos¹⁶ va generando cada uno de los posibles subconjuntos de la/s clave/s del esquema de relación, calculando su cierre transitivo y comprobando si en el cierre se encuentran atributos no principales. La complejidad de este algoritmo es exponencial a la longitud de cada una de las claves del esquema de relación.

Sea m el número de atributos de una posible clave del esquema, el número máximo de subconjuntos (y de cierres transitivos que habría que ejecutar) sería:

$$NSUB = \sum_{k=1..m-1} C_{m,k} = \sum_{k=1..m-1} (m! / (k! * (m-k)!))$$

siendo $C_{m,k}$ las combinaciones sin repetición de m elementos tomados de k en k . Por ejemplo, si la clave que se está comprobando está formada por cuatro atributos, el número máximo de subconjuntos será:

$$C_{4,3} + C_{4,2} + C_{4,1} = 4 + 6 + 4 = 14 \text{ subconjuntos (4 de longitud 3, 6 de longitud 2 y 4 de longitud 1).}$$

Con este ejemplo, en el caso peor, habría que ejecutar 14 cierres transitivos si en ninguno de los cierres anteriores hubieran aparecido atributos no principales.

Cuando en el esquema de relación existen varias claves candidatas, y suponiendo que todas las claves del esquema son disjuntas, el número máximo de subconjuntos sería:

$$\sum_{i=1..n} NSUB_i$$

n = nº claves del esquema

siendo este número inferior cuando las claves del esquema están solapadas (comparten atributos).

En el algoritmo que a continuación se describe, se realiza primero una serie de comprobaciones (fáciles de computar) que determinan automáticamente si el esquema está o no en 2FN. En el caso de que con ninguna de estas comprobaciones se pueda determinar si el esquema está o no en 2FN, se van generando y analizando cada uno de los posibles subconjuntos de cada una de las claves candidatas.

¹⁶ Este algoritmo aparece en el trabajo de fin de carrera LÓPEZ (1998) ya citado.

Las comprobaciones previas son las siguientes:

- Si existe algún atributo que no es implicado ni implicante de ninguna dependencia, entonces el esquema no está en 2FN (paso 0 del algoritmo).
- Si el conjunto Q de atributos no principales es vacío, entonces el esquema de relación está en 2FN (paso 5 del algoritmo).
- Si todas las claves del esquema de relación tienen un solo atributo, entonces el esquema está en 2FN (paso 6 del algoritmo).

El algoritmo propuesto es el siguiente:

ALGORITMO 2FN

Entrada:

Un esquema de relación R(AT,DEP)

Salida:

N(R). Forma Normal en la que está un esquema de relación.

Proceso:

- 0) Si $(\bigcup_{i=1..n} X_i, Y_i)_{i=1..n} \subset AT$ entonces

N(R)=1.

Fin del algoritmo.

Fin_si.

- 1) DEP=Recubrimiento irredundante (DEP).
- 2) N(R)=2.
- 3) Obtener CLAVES.
- 4) Obtener los conjuntos P (atributos principales) y Q (atributos no principales).
- 5) Si $Q = \emptyset$ entonces

Fin del algoritmo.

Fin_si.

- 6) Si (número de atributos de cada una de las claves es 1) entonces

Fin del algoritmo.

Fin_si.

- 7) Para cada clave C del esquema (con n° de atributos > 1)¹⁷

$k = n^\circ$ atributos de la clave C.

Para $j = k-1 \dots 1$ hacer

Mientras hay subconjuntos de C de longitud j

Generar SUBj (subconjunto de longitud j).

$\text{CIERRE} = \text{SUBj}^+_{\text{DEP}}$.

Si $(\text{CIERRE} \cap Q \neq \emptyset)$ entonces

$N(R) = 1$.

Fin del algoritmo.

Fin_si.

Fin_mientras.

Fin_para.

Fin_para_cada_clave.

Veamos la salida del algoritmo con el esquema de relación R(AT,DEP) donde:

$\text{AT} = \{A, F, B, G, C, H, D, I, E\}$

$\text{DEP} = \{A \rightarrow F, F \rightarrow A, B \rightarrow G, G \rightarrow B, C \rightarrow H, H \rightarrow C, D \rightarrow I, I \rightarrow D, \\ A B C D \rightarrow E\}$

El paso 0 no se cumple, ya que todos los atributos pertenecen a un implicado o a un implicante, por lo tanto, el algoritmo continúa.

¹⁷ Cuando existen descriptores principales equivalentes ($X_i \rightarrow Y_i$ e $Y_i \rightarrow X_i$) no es necesario hacer estas comprobaciones para todas las claves, sino únicamente para aquellas de las que forma parte X_i (o Y_i). Sin embargo, en la actual implementación del algoritmo no se ha tenido en cuenta esta condición, ni tampoco alguna otra que puede mejorar la eficiencia del algoritmo.

En el paso 1 se calcula el recubrimiento irredundante del conjunto DEP, la salida es el propio conjunto DEP.

Del paso 3 se obtienen 16 claves candidatas de este esquema, que son:

CLAVES= { ABCD, ABCI, ABHD, ABHI, AGCD, AGCI, AGHD, AGHI,
FBCD, FBCI, FBHD, FBHI, FGCD, FGCI, FGHD, FGHI }

En el paso 4 se calculan los conjuntos P y Q que son:

$P = \{A, F, B, G, C, H, D, I\}$

$Q = \{E\}$

La condición impuesta en el paso 5 no se cumple, el conjunto Q de atributos no principales es no vacío, por lo tanto, el algoritmo continúa.

La condición impuesta en el paso 6 tampoco se cumple, las claves candidatas del esquema de relación tienen más de un atributo, por lo tanto, el algoritmo continúa.

En el paso 7 se generan los subconjuntos de cada clave. Para la primera de ellas, habría que calcular 14 cierres transitivos ($C_{4,3} + C_{4,2} + C_{4,1}$). Para el resto de las claves, y puesto que las claves están solapadas, no hay que calcular los 14 cierres por cada una de ellas, puesto que algunos de ellos ya habrán sido generados y comprobados con la clave anterior¹⁸.

No se encuentra ningún subconjunto de las claves en cuyo cierre aparezcan atributos no principales.

El algoritmo finaliza y el esquema de relación evaluado está en 2FN.

4.2. Determinación de 3FN

Para presentar este algoritmo utilizamos la definición de 3FN dada por ULLMAN (1982).

Definición:

Un esquema de relación $R(AT, DEP)$ está en 3FN si, para cualquier dependencia trivial $X \rightarrow Y$ de DEP, se cumple que X es una superclave o el atributo Y es principal.

¹⁸ En realidad sólo sería necesario hacer las comprobaciones para los subconjuntos de una única clave, ya que el resto de los atributos principales son equivalentes a los que forman parte de la clave; incluso, con un tratamiento completo de atributos equivalentes, en este caso no habría habido necesidad de hallar ningún cierre.

Según esta definición, si una dependencia $X \rightarrow Y$ (donde Y es no principal) de DEP viola la definición de 3FN, ocurrirá uno de los dos siguientes casos:

1. X es un subconjunto propio de la clave.
2. X es un subconjunto propio de un conjunto no clave.

En el primer caso, la dependencia $X \rightarrow Y$ será una dependencia parcial, violando así la definición de 2FN, ya que existe un atributo no principal que depende de parte de la clave.

En el segundo caso, la dependencia $X \rightarrow Y$ será una dependencia transitiva, ya que si C es la clave del esquema, encontraremos la siguiente cadena de dependencias $C \rightarrow X \rightarrow Y$.

El algoritmo que a continuación proponemos está basado en la definición anterior. La diferencia principal respecto a otros algoritmos está en que aquí se va comprobando simultáneamente si se cumplen las condiciones que definen la segunda y tercera forma normal, mientras que en otros algoritmos primero se comprueba que el esquema está en 2FN, y posteriormente se verifica que cada implicante tenga al menos un atributo principal.

El algoritmo es el siguiente:

ALGORITMO 3FN¹⁹

Entrada:

Un esquema de relación $R(AT, DEP)$

Salida:

“Sí” o “No” para indicar si el esquema está o no en 3FN.

Proceso:

- 1) $DEP = \text{Recubrimiento_irredundante}$
- 2) Calcular claves de $R(AT, DEP)$

¹⁹ Este algoritmo podría mejorarse incluyendo algunas comprobaciones previas, así como introduciendo un cambio para que, en lugar de obtener como salida “Sí” o “No”, indicará si el esquema está en 2FN o 3FN; sin embargo, hemos preferido presentar el algoritmo tal como se encuentra implementado en la actual versión de RENO que se incluye en esta obra.

3) P = conjunto de atributos principales.

Q = conjunto de atributos no principales.

RESULTADO=SI

4) Para toda $X \rightarrow Y \in \text{DEP}$

Si X no es superclave entonces

Si $A \in Q$ entonces

RESULTADO=NO.

Fin del algoritmo.

Fin_si.

Fin_si.

Fin_para_todo.

Veamos la salida del algoritmo con el esquema de relación $R(AT, DEP)$ donde:

$AT = \{A, B, C, D, E\}$

$DEP = \{AB \rightarrow C, A \rightarrow D, D \rightarrow A, C \rightarrow E\}$

En el paso 1 se calcula el recubrimiento irredundante del conjunto DEP , la salida es el propio conjunto DEP .

En el paso 2 se calculan las claves del esquema, que son: AB y DB .

En el paso 3 se calculan los conjuntos de atributos principales y no principales, que son: $P = \{A, B, D\}$ y $Q = \{C, E\}$

Analizando cada una de las dependencias del esquema, vemos que para la dependencia $C \rightarrow E$, se cumple que el implicante C no es clave (o superclave) y el atributo E es no principal. Por lo tanto, se encuentra una dependencia que viola la definición de 3FN. El esquema no está en 3FN.

4.3. Determinación de FNBC

Definición:

Un esquema de relación $R(AT, DEP)$ está en Forma Normal de Boyce_Codd si, y sólo si, todo determinante es una clave candidata del esquema de relación $R(AT, DEP)$.

Por lo tanto, en un esquema de relación en FNBC las únicas dependencias no triviales son aquellas en las que los implicantes determinan funcionalmente todos los atributos AT .

Un algoritmo para determinar si un esquema de relación está en FNBC, aparece descrito en Ullman (1982).

La complejidad de dicho algoritmo viene determinada, por un lado, por la complejidad del cálculo de las claves y, una vez calculadas dichas claves, es polinomial al número de dependencias del esquema.

Por la propia definición de la clave, dado $R(AT, DEP)$, un descriptor X es clave de R si y sólo si $X^+_{DEP} = AT$. Utilizando esta definición se puede comprobar si un esquema de relación está en FNBC sin necesidad de calcular las claves, evitando el tiempo de cálculo que consume esta operación. El algoritmo propuesto por MANILA (1992), es el siguiente:

ALGORITMO FNBC

Entrada:

Un esquema de relación $R(AT, DEP)$.

Salida:

Una decisión sobre si el esquema de relación está o no en FNBC.

Proceso:

0) $DEP =$ Recubrimiento irredundante.

1) $RESULTADO = 'SI'$.

1) Para toda $X \rightarrow Y$ de DEP

$CIERRE = X^+_{DEP}$.

Si $(\text{CIERRE} \cap \text{AT}) \neq \text{AT}$ entonces

RESULTADO='NO'.

Fin del algoritmo.

Fin_si.

Fin_para_toda.

La complejidad de este algoritmo es polinomial al número de dependencias definidas en el esquema, puesto que con este algoritmo no es necesario calcular las claves del esquema. Únicamente se va calculando el cierre transitivo de cada uno de los determinantes.

El algoritmo **finalizará** en cuanto se encuentre un implicante en cuyo cierre transitivo no estén todos los **atributos** del esquema (el esquema no está en FNBC), o cuando se compruebe que los **cierres** de todos los implicantes contienen todos los atributos del esquema (el esquema está en FNBC).

Veamos la salida del algoritmo con el esquema de relación $R(\text{AT}, \text{DEP})$ donde:

$\text{AT} = \{A, F, B, G, C, H, D, I, E\}$

$\text{DEP} = \{A \rightarrow F, F \rightarrow A, B \rightarrow G, G \rightarrow B, C \rightarrow H, H \rightarrow C, D \rightarrow I, I \rightarrow D, \\ ABCD \rightarrow E\}$

En el paso 0, se calcula el recubrimiento irredundante del conjunto DEP, la salida es el propio conjunto DEP.

El paso 2 comienza, por ejemplo, con la dependencia $A \rightarrow F$, vemos que el cierre transitivo del atributo A no contiene todos los atributos del esquema, el algoritmo finaliza (consumiendo sólo el tiempo necesario para calcular un cierre transitivo), concluyendo que el esquema analizado no se encuentra en FNBC.

Este algoritmo es mucho más rápido que el propuesto por Ullman, puesto que se evita realizar el cálculo de las claves del esquema de relación (este cálculo puede penalizar mucho el tiempo de respuesta del algoritmo). En LÓPEZ (1998), se realiza un análisis de los tiempos de respuesta de ambos algoritmos, demostrando que, en todos los conjuntos de dependencia de prueba, los tiempos de este algoritmo son siempre mucho menores.

5. ALGORITMOS DE DESCOMPOSICIÓN DE UN ESQUEMA DE RELACIÓN EN ESQUEMAS EN FNBC

Los esquemas de relación en 3FN evitan redundancias y anomalías de inserción y borrado. Aun así, existen algunas relaciones en 3FN que siguen presentando anomalías. Esto ocurre cuando en el esquema de relación existen claves candidatas que se solapan.

Sea el esquema de relación $R(AT, DEP)$ donde:

$AT = \{A, B, C\}$

$DEP = \{AB \rightarrow C, C \rightarrow A\}$

Las claves candidatas del esquema son: (AB) y (CB) . Estas claves se solapan puesto que comparten el atributo común B . El esquema de relación está en 3FN. El problema que presenta este esquema es que no se puede almacenar en la base de datos la pareja $(c1, a1)$ hasta que no se conozca el valor $b1$ del atributo B asociado a $c1$. Para evitar esta anomalía, se hace necesario normalizar hasta FNBC.

5.1. Descomposición en esquemas FNBC

El algoritmo que a continuación se describe analiza el esquema de relación comprobando si existen dependencias que violan la definición de FNBC, y si existe(n) tal(es) dependencia(s), divide el esquema en dos proyecciones independientes. Si, a su vez, en los esquemas generados existen dependencias que violan la FNBC, se vuelve a dividir el esquema en dos nuevos esquemas. Este proceso se repite iterativamente y finaliza cuando todos los esquemas resultantes están en FNBC.

El algoritmo de normalización de esquemas en FNBC, propuesto por Manila (1992), es el siguiente:

ALGORITMO DESC_1

Entrada:

Esquema de relación $R=(AT, DEP)$.

Salida:

$ESQ = R1..Rn$ conjunto de esquemas de relación en FNBC.

Proceso:

- 0) Si (R está en FNBC) entonces

ESQ= { R }.

Fin del algoritmo.

Fin_si.

- 1) ESQ = { }.
- 2) DEP = recubrimiento_irredundante(DEP).

Calcular las claves del esquema DEP.

- 3) Mientras (existan dependencias $X \rightarrow Y \in \text{DEP}$ | X no es superclave:

3.1 $\text{AT}_i = X \cup Y$

3.2 $\text{AT} = \text{AT} - Y$

3.3 $\text{PR}_i = \text{Proyección de DEP sobre AT}_i$

3.4 $\text{DEP} = \text{Proyección de DEP sobre AT}$

3.5 $\text{R}_i = (\text{AT}_i, \text{PR}_i)$

3.6 $\text{ESQ} = \text{ESQ} \cup \{ \text{R}_i \}$

Fin_mientras

El algoritmo finalizará cuando todos los esquemas generados estén en FNBC. Puede observarse que en el caso de que existan varias dependencias que violan la FNBC (paso 3), el algoritmo no dice nada acerca de cuál es la dependencia que se debe elegir y, por tanto, se puede elegir cualquiera. Aunque a simple vista este aspecto no parece importante, veremos que según cual sea la dependencia elegida en cada momento, la descomposición final será distinta, siendo en unos casos mejor que en otros.

Antes de aplicar el algoritmo a un esquema de relación ejemplo, vamos a definir el concepto de proyección de un conjunto de dependencias sobre un conjunto de atributos.

5.2. Proyección de un conjunto de dependencias sobre un conjunto de atributos

Es importante tener en cuenta que, para definir completamente cada uno de los esquemas resultantes de la descomposición, es necesario conocer, además de los atributos que forman parte del nuevo esquema de relación, las dependencias que existen entre los atributos de este nuevo esquema. Este conjunto de dependencias se conoce como la **proyección del conjunto DEP de dependencias de partida sobre el conjunto de atributos X del nuevo esquema** (pasos 3.3 y 3.4 del algoritmo anteriormente descrito), y se define formalmente como sigue:

Definición:

La proyección de un conjunto DEP de dependencias funcionales sobre un conjunto de atributos X, denotado como $DEP[X]$ es el conjunto de dependencias $Y \rightarrow Z$ de DEP para las que se cumple que $YZ \subseteq X$. Es decir:

$$DEP[X] = \{ Y \rightarrow Z \mid DEP = Y \rightarrow Z \text{ y } YZ \subseteq X \}$$

Puede ocurrir que $Y \rightarrow Z$ no esté en DEP, es suficiente con que $Y \rightarrow Z$ esté en DEP^+ .

El algoritmo que calcula la proyección de un conjunto de dependencias sobre un conjunto de atributos, propuesto por MANILA (1992), es el siguiente:

ALGORITMO PROY

Entrada:

Un conjunto DEP de dependencias funcionales sobre un conjunto AT de atributos.

Un subconjunto $X \subseteq AT$.

Salida:

Un recubrimiento de $DEP[X]$

Proceso:

- 1) $G =$ Recubrimiento irredundante (DEP).
- 2) $W = AT - X$.

3) Mientras ($W \neq \emptyset$) hacer:

3.1) $A =$ cualquier atributo de W .

3.2) $W = W - A$.

3.3) $H = \{ ZY \rightarrow B \mid \text{las dependencias } Z, A \rightarrow B \text{ y } Y \rightarrow A \text{ están en } G \}$.

3.4) Eliminar las dependencias triviales de H .

3.5) $G = G - \{ f \in G \mid A \text{ ocurre en } f \} \cup H$.

Fin_mientras.

4) $DEP[X] = G$.

Veamos la salida del algoritmo con el esquema de relación $R(AT, DEP)$ donde:

$AT = \{C, D, E, I, J, K, L\}$

$DEP = \{C \rightarrow D, E \rightarrow J, I \rightarrow J, J \rightarrow K, K \rightarrow J, DK \rightarrow L\}$

La proyección de DEP sobre $X = \{C, E, I, L\}$ se calcula como sigue:

1) $G = \{C \rightarrow D, E \rightarrow J, I \rightarrow J, J \rightarrow K, K \rightarrow J, DK \rightarrow L\}$

2) $W = \{D, J, K\}$

3) Mientras ($W \neq \emptyset$)

1ª Iteración

$A = D$

$W = \{J, K\}$

$H = \{CK \rightarrow L\}$

$G = G - \{C \rightarrow D, DK \rightarrow L\} \cup \{CK \rightarrow L\}$

2ª Iteración

$A = J$

$W = K$

$H = \{I \rightarrow K, E \rightarrow K\}$

$G = G - \{E \rightarrow J, I \rightarrow J, K \rightarrow J, J \rightarrow K\} \cup \{I \rightarrow K, E \rightarrow K\}$

3ª Iteración

$$A = K$$

$$W = \{ \}$$

$$H = \{ CI \rightarrow L, CE \rightarrow L \}$$

$$G = G - \{ CK \rightarrow L, I \rightarrow K, E \rightarrow K \} \cup \{ CI \rightarrow L, CE \rightarrow L \}$$

$$4) \text{ DEP } [C, E, I, L] = \{ CE \rightarrow L, CI \rightarrow L \}$$

El algoritmo finaliza después de $|AT| - |X|$ iteraciones.

5.3. Algoritmo de descomposición en esquemas FNBC que mejora la eficiencia

MANILA (1992) propone una mejora en la eficiencia del algoritmo consistente en considerar del conjunto de dependencias DEP sólo aquellas que realmente intervienen en el cálculo de la proyección. La determinación de este conjunto está basado en el concepto de antecedentes de un descriptor.

Definición:

Sea el esquema de relación $R(AT, DEP)$ y sea $X \subseteq AT$. Se definen los **antecedentes** de X y se denota como X_{DEP}^- , el conjunto de atributos de AT , que pueden ser usados para derivar una dependencia donde algún conjunto de estos atributos determina un atributo de X .

El cálculo de este conjunto se define iterativamente de la siguiente forma:

$$1) \quad X_0^- = X$$

$$2) \quad X_{i+1}^- = X_i^- \cup \{ B \mid B \in Y \text{ para cualquier } Y \rightarrow Z \in DEP \text{ para el que se cumple que } Z \cap X_i^- \neq \emptyset \}$$

El cálculo finaliza después de un número finito de iteraciones, cuando se cumpla que $X_{i+1}^- = X_i^-$.

Sea el esquema de relación $R(AT, DEP)$ donde:

$$AT = \{ A, B, C, D, E, F \}$$

$$DEP = \{ A \rightarrow B, C \rightarrow D, B, D \rightarrow E, E \rightarrow F \}.$$

El cálculo de los antecedentes del atributo E (E^-) será:

$$E_0^- = \{E\}$$

$$E_1^- = \{E\} \cup \{B, D\}$$

$$E_2^- = \{E, B, D\} \cup \{A, C\}$$

$$E_3^- = E_2^- = \{E, B, D, A, C\}$$

Basado en el concepto de antecedentes de un descriptor, se puede afirmar que para calcular un recubrimiento de **DEP[X]** **basta con considerar sólo** las dependencias de DEP de la forma $Y \rightarrow B$ tales que para algún Z con $B \in Z$ y $Y \rightarrow Z \in \text{DEP}$, se cumple que $YB \subseteq X^+_{\text{DEP}} \cap X^-_{\text{DEP}}$.

Por lo tanto en el algoritmo PROY se incluye un paso inicial, que es la selección del conjunto de dependencias que realmente interviene en el cálculo de la proyección, evitando así comprobar las dependencias que no aportan nada al cálculo. Éste es el algoritmo que se ha implementado en RENO como base del algoritmo de descomposición a esquemas en FNBC.

Volviendo al algoritmo de descomposición DESC_1, es importante indicar que no siempre se puede asegurar que la descomposición preserva las dependencias, aunque sí se puede asegurar que la descomposición se realiza sin pérdida de información. Sea $X \rightarrow Y$ una dependencia del esquema R_k , que viola la FNBC, descomponemos R_k en R_i y R_j donde:

$$AT_i = AT_k - Y$$

$$AT_j = X, Y$$

Aplicando la condición de Rissanen (el descriptor común es clave en alguna de las proyecciones), se comprueba que $AT_i \cap AT_j = X$, $AT_j - AT_i = Y$ y la dependencia $X \rightarrow Y$ está en AT_j .

Veamos cómo trabajaría el algoritmo con el esquema de relación $R(AT, \text{DEP})$ donde:

$$AT = \{A, B, C, D, E, F, X\}$$

$$\text{DEP} = \{A \rightarrow X, X \rightarrow A, AB \rightarrow C, AB \rightarrow D, D \rightarrow E, E \rightarrow F\}.$$

En el paso 2 del algoritmo se calcula el recubrimiento irredundante del conjunto DEP, el resultado es el propio conjunto DEP. Las claves de este esquema son AB y BX.

Observamos que en este esquema existen varias dependencias cuyo implicante no es **superclave**. Éstas son: $A \rightarrow X$, $X \rightarrow A$, $D \rightarrow E$ y $E \rightarrow F$.

Podríamos comenzar la descomposición por cualquiera de ellas, pues la única **restricción** que impone el algoritmo es que el implicante de la dependencia elegida no sea **superclave** del esquema. Comencemos, por ejemplo, con la dependencia $D \rightarrow E$. Descomponemos el esquema R en los esquemas:

R1 (AT1, DEP1)	R2 (AT2, DEP2)
AT1 = { D, E }	AT2 = { A, B, C, D, F, X }
DEP1 = { $D \rightarrow E$ }	DEP2 = { $A \rightarrow X$, $X \rightarrow A$, $AB \rightarrow C$, $AB \rightarrow D$, $D \rightarrow F$ }

donde DEP1 y DEP2 son las proyecciones de DEP sobre AT1 y AT2 respectivamente.

El esquema R1 ya está en FNBC, pero el esquema R2 no (existen descriptores que no son claves del esquema), por lo tanto continúa el proceso de descomposición. Al igual que el caso anterior, podríamos elegir cualquiera de las dependencias que violan la FNBC ($A \rightarrow X$, $X \rightarrow A$, $D \rightarrow F$) elegimos, por ejemplo, $D \rightarrow F$. El esquema R2 se descompone en los esquemas:

R3 (AT3, DEP3)	R4 (AT4, DEP4)
AT3 = { D, F }	AT4 = { A, B, C, D, X }
DEP3 = { $D \rightarrow F$ }	DEP4 = { $A \rightarrow X$, $X \rightarrow A$, $AB \rightarrow C$, $AB \rightarrow D$ }

donde DEP3 y DEP4 son las proyecciones de DEP2 sobre AT3 y AT4 respectivamente.

El esquema R3 ya está en FNBC, pero el esquema R4 no (existen descriptores que no son claves del esquema), por tanto, continúa el proceso de descomposición. En este esquema podríamos elegir entre la dependencia $A \rightarrow X$ o $X \rightarrow A$. Elegimos $A \rightarrow X$, el esquema R4 se descompone en:

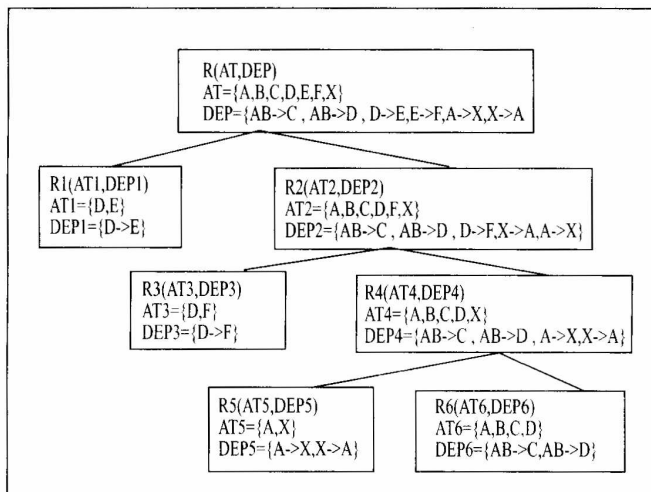
R5(AT5, DEP5)	R6(AT6, DEP6)
AT5 = { A, X }	AT6 = { A, B, C, D }
DEP5 = { $A \rightarrow X$, $X \rightarrow A$ }	DEP6 = { $AB \rightarrow C$, $AB \rightarrow D$ }

donde DEP5 y DEP6 son las proyecciones de DEP4 sobre AT5 y AT6 respectivamente.

Tanto R5 como R6 están en FNBC. El algoritmo finaliza y la descomposición resultante es:

R1, R3, R5, R6.

El proceso de descomposición se puede ver gráficamente como un árbol binario cuya raíz es el esquema de partida, y cada una de sus hojas forman el conjunto de esquemas en FNBC resultantes de la descomposición. Para este ejemplo, el árbol de descomposición es el siguiente:



La descomposición es sin pérdida de información (SPI), pero la dependencia $E \rightarrow F$ se ha perdido en el proceso de descomposición. Un recubrimiento irredundante de la unión de los esquemas resultantes de la descomposición es:

$$\text{REC} = \{D \rightarrow E, D \rightarrow F, A \rightarrow X, X \rightarrow A, AB \rightarrow C, AB \rightarrow D\}$$

y aplicando los axiomas de Armstrong no se obtiene que $(\text{REC})^+ \models E \rightarrow F$

Cabe preguntarse si no existe ninguna otra descomposición del esquema de partida en la que no se produzca pérdida de dependencias. Para este caso en concreto, la respuesta es afirmativa, y es que tanto las dependencias elegidas en la descomposición como el orden con el que estas dependencias son elegidas puede provocar distintas descomposiciones finales. En la versión mejorada de este algoritmo (véase algoritmo DESC_2) veremos que, para el esquema de relación que hemos utilizado como ejemplo, existe otra descomposición en la que no se produce pérdida de dependencias.

Por supuesto, existen otros casos en los que en la descomposición final se producirá pérdida de dependencias sea cual sea el orden en el que se toman las dependencias en el proceso de descomposición.

Por último recordar que el objetivo del algoritmo es conseguir esquemas en FNBC aunque se produzca pérdida de dependencias. En la herramienta RENO se ha

implementado el algoritmo tal y como se ha descrito, incorporando al final del mismo una llamada al algoritmo de preservación de dependencias, presentando en pantalla las dependencias que se han perdido en la descomposición (si las hubiera). Finalmente, también se presenta una pantalla con las dependencias en el mismo orden en que se han elegido en cada fase del proceso de descomposición.

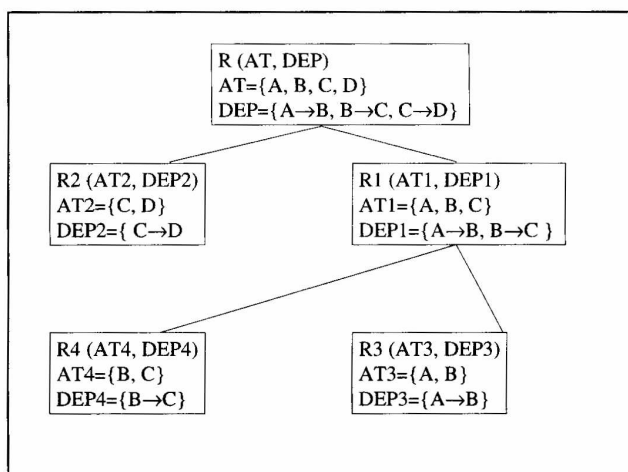
5.4. Nuevo algoritmo de descomposición en esquemas FNBC que mejora la funcionalidad²⁰

Cuando en el proceso de descomposición existen varias dependencias que violan la FNBC, el orden en el que se toman las dependencias para realizar la descomposición influye en el resultado final de la misma, generándose así distintas descomposiciones, unas mejores que otras, tanto desde el punto de vista del número de esquemas resultantes y contenido de los mismos como desde el punto de vista semántico (qué dependencias quedan recogidas en la nueva descomposición).

Veámoslo con un sencillo ejemplo. Sea $R(AT, DEP)$ un esquema de relación donde $AT = \{A, B, C, D\}$ y $DEP = \{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$.

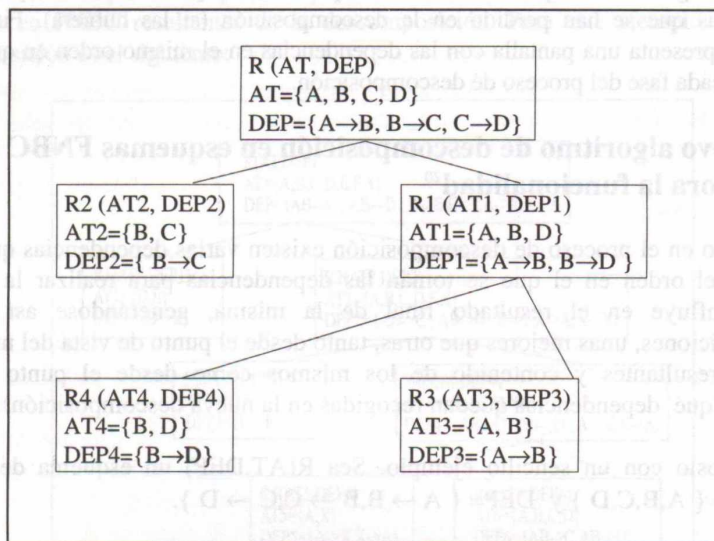
La única clave es A. Tanto la dependencia $C \rightarrow D$ como la dependencia $B \rightarrow C$, violan la FNBC, puesto que ninguno de los dos determinantes son claves. Para realizar la descomposición podríamos elegir tanto la dependencia $B \rightarrow C$ como la dependencia $C \rightarrow D$.

El árbol de descomposición, comenzando por la dependencia $C \rightarrow D$ es el siguiente:



²⁰ Este algoritmo ha sido propuesto en LÓPEZ (1998).

El árbol de descomposición comenzando por la dependencia $B \rightarrow C$ es el siguiente:



Ambas descomposiciones generan tres esquemas en FNBC, ¿por cuál de ellas optamos? El algoritmo no puede determinarlo.

En este caso es importante conocer la semántica de los datos, pues dependiendo de la información que cada nuevo esquema almacene, se optará por la descomposición que refleje más fiel y coherentemente la información que queremos almacenar en la base de datos; pero, para considerar la semántica, sería necesaria la intervención del usuario.

Aparte de las consideraciones semánticas, es muy importante destacar que el orden en que se elijen las dependencias puede provocar, en algunos casos, que en la descomposición se produzca o no pérdida de las mismas.

Siguiendo con el ejemplo anterior, se observa que en la primera de las descomposiciones no se produce pérdida de dependencias, la unión de las dependencias de cada uno de los esquemas es un recubrimiento del conjunto DEP de dependencias de partida. En cambio, en la segunda descomposición sí se produce la pérdida de la dependencia $C \rightarrow D$.

Es muy importante realizar la descomposición de forma que se conserven todas las dependencias funcionales posibles. Por tanto, no basta con seleccionar cualquiera de las dependencias cuyo implicante no sea superclave (tal y como indica el algoritmo DESC_1), hay que imponer alguna condición más. Esta versión mejorada aborda el problema de seleccionar en todo momento la dependencia que mejor descomposición

final produce (mejor descomposición en el sentido de que no se pierdan dependencias).

Hay que tener en cuenta que existen esquemas de relación para los cuales, debido a su estructura, **no es posible** encontrar una descomposición a esquemas en FNBC sin que se produzca pérdida de semántica: existe en cambio otro conjunto, que es el que nos ocupa, en el que unas descomposiciones pierden semántica y otras no, dependiendo de las dependencias que se van seleccionando durante el proceso de descomposición.

Una posible opción sería que el diseñador fuera "guiando" al algoritmo, indicando, en cada paso del proceso de descomposición, la dependencia a utilizar. Esta posibilidad no ha sido considerada porque la herramienta RENO está orientada tanto a usuarios familiarizados con el diseño de BD relacionales como a los que no lo están.

La pérdida de dependencias se produce al desaparecer del esquema un atributo que se encuentra en alguna de las dependencias y, por tanto, ésta desaparece sin que sea posible deducirla del resto. En cada paso del algoritmo y, una vez seleccionada la dependencia por la que descomponer (por ejemplo $X \rightarrow Y$), se generan dos esquemas: el primero está formado por los atributos X e Y , y el segundo está formado por los mismos atributos que había menos el atributo Y . En el cálculo de la proyección sobre este segundo esquema, se transforman aquellas dependencias que contengan el atributo que no se encuentra ya en el conjunto AT (es decir, el atributo Y) en dependencias cuyos atributos sí se encuentren en dicho conjunto, aplicando para ello, las reglas de inferencia de dependencias.

Los criterios que vamos a imponer para evitar que se produzca pérdida de semántica en una descomposición afectan a dos grupos de dependencias. Éstos son:

1) Dependencias que definen equivalencia entre atributos no principales

Sea $R(AT, DEP)$ un esquema de relación

$AT = \{CT, CA, NA, DT, E, I, O, HO\}$

$DEP = \{ CT \rightarrow CA, CA \rightarrow CT, CT \rightarrow NA, NA \rightarrow CT, CT \rightarrow DT, DT \rightarrow CT, E \rightarrow NA, I \rightarrow O, EI \rightarrow HO, A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow C \}$

Los atributos equivalentes son CT , CA , NA y DT . Dichos atributos son no principales, pues la clave del esquema es I, D, E y además son implicados en otras dependencias.

Si elegimos por pasos sucesivos, las dependencias que definen la equivalencia de atributos (por ejemplo $CT \rightarrow CA$, $CT \rightarrow NA$, $CT \rightarrow DT$) durante cada fase del proceso de descomposición, el cálculo de la proyección irá transformando las dependencias que

tienen el atributo que es el implicado de la dependencia por la que se descompone, sustituyendo para ello dicho atributo por cualquiera de los atributos equivalentes a éste y que aún se encuentren en el esquema de relación (es decir, con la primera dependencia se transformarán las dependencias que tienen el atributo CA, luego las que tienen el atributo CT y así sucesivamente).

Al realizar la descomposición por la última de estas dependencias ($CT \rightarrow DT$), nos encontraremos con que en el esquema todas las dependencias cuyo implicado era alguno de estos atributos equivalentes, tendrán ahora el mismo atributo como implicado, justo el implicado de la dependencia por la que se va a descomponer (atributo DT), el cual desaparecerá del esquema al realizar el cálculo de la proyección, con lo que se perderán aquellas dependencias en el cual aparecía este atributo.

Para solucionar este problema, y utilizando la equivalencia de atributos, lo que proponemos es, al seleccionar una dependencia que defina atributos equivalentes, incluir en el nuevo esquema que se genera no sólo los atributos de la dependencia por la que se descompone, sino todos aquellos que sean equivalentes; de esta forma, el cálculo de la proyección incluirá en un solo paso todas las dependencias que definen atributos equivalentes y evitará que en el esquema restante se vayan transformando las dependencias y acaben teniendo varias de ellas el mismo atributo implicado.

De este modo, al seleccionar la primera de estas dependencias $CT \rightarrow CA$, el nuevo esquema que se genera estaría formado por los atributos CT y CA y por los equivalentes a estos, es decir, DT y NA, y al calcular la proyección sobre este conjunto de atributos, las dependencias que formarían este nuevo esquema serían: $\{CT \rightarrow CA, CA \rightarrow CT, CT \rightarrow NA, NA \rightarrow CT, CT \rightarrow DT, DT \rightarrow CT\}$.

2) Dependencias cuyos implicados no son implicantes de otra dependencias

Supongamos que tenemos las dependencias $X \rightarrow Y$ e $Y \rightarrow Z$ y que Z no es implicante de otra dependencia.

En este caso, el atributo Y actúa como implicante e implicado, si eligieramos la dependencia $X \rightarrow Y$, el atributo Y desaparecería del esquema original y la dependencia $Y \rightarrow Z$ tendría que ser transformada durante el cálculo de la proyección.

En el caso contrario, si eligieramos la dependencia $Y \rightarrow Z$, Z desaparecería del esquema original (pues pasa a formar parte del nuevo esquema que se genera) pero la dependencia $X \rightarrow Y$ no sufre ninguna transformación durante el cálculo de la proyección.

Por tanto, siempre que sea posible, las primeras dependencias que se elegirán en el proceso de descomposición serán aquellas cuyo implicado no sea implicante de otra dependencia.

El algoritmo que selecciona la dependencia por las que se realiza la descomposición es el siguiente:

ALGORITMO SELECCIONA

Entrada:

Un conjunto DEP de dependencias funcionales que violan la FNBC.

Salida:

D dependencia seleccionada.

TIPO Tipo de la dependencia seleccionada.

Proceso:

0) CONJUNTO 1 = Dependencias que definen atributos equivalentes no principales.

1) Si $\text{CONJUNTO1} \neq \emptyset$ entonces

D = Cualquier dependencia de CONJUNTO1

TIPO = 1

Fin_algoritmo

2) $\text{IMPLICANTES} = \emptyset$

3) Para toda $X \rightarrow Y$ de DEP

Implicantes=Implicantes U X

Fin_para_toda

4) $\text{CANDIDATAS} = \emptyset$

Para toda $X \rightarrow Y$ de DEP

Si Y no pertenece a Implicantes entonces

$\text{CANDIDATAS} = \text{CANDIDATAS} \cup (X \rightarrow Y)$

Fin_para_toda

5) Si $CANDIDATAS \neq \emptyset$ entonces

D= Cualquier dependencia de CANDIDATAS

En caso contrario

D= Cualquier dependencia de DEP

Fin_si

La primera parte del algoritmo localiza las dependencias que definen equivalencia entre atributos no implicados. La segunda parte genera una lista con los atributos que son implicantes, y para cada una de las dependencias que violan la FNBC, comprueba si el implicado es implicante en alguna otra dependencia. Si esto no ocurre, esta dependencia es una posible candidata, para ser seleccionada como la dependencia por la que realizar la descomposición. En el caso de que no se encuentre ninguna dependencia cuyo implicado no sea implicante de ninguna otra dependencia, el algoritmo elegirá cualquier dependencia del conjunto DEP de entrada, igual que haría la version tradicional del algoritmo DESC_1

El algoritmo de descomposición de esquemas en FNBC, DESC_1, seleccionando previamente la dependencia por la que realizar la descomposición, queda como sigue:

ALGORITMO DESC_2

Entrada:

Esquema de relación $R=(AT,DEP)$

Salida:

ESQUEMAS = $R1..Rn$ conjunto de esquemas de relación en FNBC.

Proceso:

0) Si R está en FNBC entonces

ESQ= { R }

fin del algoritmo

Fin_si

1) ESQ = { }

2) DEP = Recubrimiento irredundante (DEP)

Calcular las claves del esquema DEP

3) Mientras existan dependencias $X \rightarrow Y \in \text{DEP}$ | X no es superclave

3.1) $X \rightarrow Y$ = Salida del algoritmo SELECCIONA

3.2) Si (TIPO=1) entonces

$$AT_i = X \cup Y \cup \{\text{Atributos equivalentes}\}$$

En caso contrario

$$AT_i = X \cup Y$$

Fin_si

3.3) $AT = AT - X$

3.4) PR_i = Proyección de DEP sobre AT_i

3.5) DEP' = Proyección de DEP sobre AT

3.6) $R_i = (AT_i, PR_i)$

3.7) $\text{ESQ} = \text{ESQ} \cup \{ R_i \}$

Fin_mientras

4) ESQUEMAS = ESQ

Veamos cómo trabajaría el algoritmo con el esquema de relación $R(AT, \text{DEP})$ donde:

$$AT = \{ A, B, C, D, E, F, X \}$$

$$\text{DEP} = \{ A \rightarrow X \quad X \rightarrow A \quad AB \rightarrow C \quad AB \rightarrow D \quad D \rightarrow E \quad E \rightarrow F \}.$$

En el paso 2 del algoritmo se calcula el recubrimiento irredundante del conjunto DEP, el resultado es el propio conjunto DEP. Las claves de este esquema son AB y XB.

Analizando las dependencias del esquema R, vemos que las dependencias que violan la FNBC son: $E \rightarrow F$, $D \rightarrow E$, $A \rightarrow X$, $X \rightarrow A$

Aplicando el algoritmo SELECCIONA a este conjunto de dependencias, vemos que la única dependencia cuyo implicado no es implicante de ninguna dependencia es $E \rightarrow F$, por lo tanto comenzamos la descomposición por esta dependencia.

Descomponemos el esquema R en los esquemas R1 y R2:

R1 (AT1,DEP1) R2 (AT2,DEP2)

AT1= { E, F } AT2= { A,B,C,D,E,X }

DEP1= { $E \rightarrow F$ } DEP2={ $A \rightarrow X$, $X \rightarrow A$, $AB \rightarrow C$, $AB \rightarrow D$, $D \rightarrow E$ }

donde DEP1 y DEP2 son las proyecciones de DEP sobre AT1 y AT2 respectivamente.

El esquema R1 ya está en FNBC, pero el esquema R2 no (existen descriptores que no son claves del esquema), por lo tanto continúa el proceso de descomposición. En este esquema R2, las dependencias que violan la FNBC son: $A \rightarrow X$, $X \rightarrow A$, $D \rightarrow E$.

Aplicando el algoritmo SELECCIONA a este conjunto de dependencias, vemos que la única dependencia cuyo implicado no es implicante de ninguna dependencia es $D \rightarrow E$, por lo tanto comenzamos la descomposición por esta dependencia.

Descomponemos el esquema R2 en los esquemas R3 y R4:

R3 (AT3,DEP3) R4 (AT4,DEP4)

AT3= { D, E } AT4= { A,B,C,D,X }

DEP3= { $D \rightarrow E$ } DEP4={ $A \rightarrow X$ $X \rightarrow A$ $A,B \rightarrow C$ $A,B \rightarrow D$ }

donde DEP3 y DEP4 son las proyecciones de DEP2 sobre AT3 y AT4 respectivamente.

El esquema R3 ya está en FNBC, pero el esquema R4 no (existen descriptores que no son claves del esquema), por lo tanto, continúa el proceso de descomposición. En este esquema encontramos que las dependencias que violan la FNBC son: $A \rightarrow X$ y $X \rightarrow A$

Aplicando el algoritmo SELECCIONA a este conjunto de dependencias, el algoritmo no encuentra ninguna dependencia cuyo implicado no es clave de ninguna dependencia. En este caso, el algoritmo elige cualquiera de las dos dependencias candidatas tal y como haría el algoritmo DESC_1. Realizamos la descomposición por la dependencia $A \rightarrow X$.

Descomponemos el esquema R4 en los esquemas R5 y R6:

$R5(AT5, DEP5)$

$R6(AT6, DEP6)$

$AT5 = \{A, X\}$

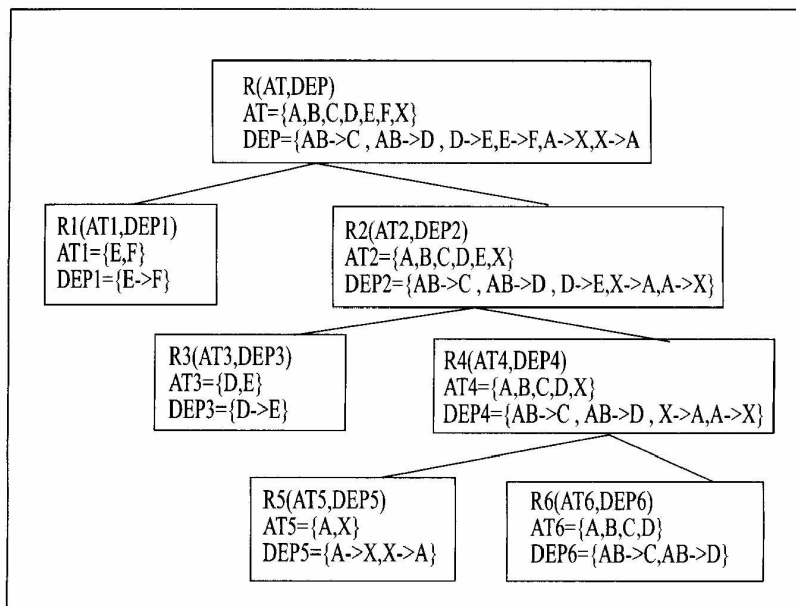
$AT6 = \{A, B, C, D\}$

$DEP5 = \{A \rightarrow X, X \rightarrow A\}$ $DEP6 = \{A, B \rightarrow C, A, B \rightarrow D\}$

donde DEP5 y DEP6 son las proyecciones de DEP4 sobre AT5 y AT6, respectivamente.

Tanto R5 como R6 ya están en FNBC. El algoritmo finaliza y la descomposición resultante es R1, R3, R5, R6.

Gráficamente, el árbol binario de descomposición es el siguiente:



La descomposición es SPI, y además preserva todas las dependencias del esquema de partida.

Por último, decir que el objetivo de este algoritmo es el de generar los esquemas resultantes de la descomposición de forma que se pierda el menor número de dependencias posible, intentando así reflejar lo más fielmente la semántica del Universo del Discurso que se pretende modelar, y por ello, para algunos conjuntos de dependencias, el tiempo de cálculo de esta versión mejorada puede ser más elevado

(puesto que en este algoritmo se realizan un cálculos adicionales para seleccionar las dependencias por las que se va a ir realizando la descomposición).

En LÓPEZ (1998), se analiza el comportamiento de ambos algoritmos, tanto desde el punto de vista del tiempo de respuesta como de la semántica de los esquemas resultantes de la descomposición.

En RENO, de forma similar al anterior algoritmo DESC_1, al final de este algoritmo se hace una llamada al algoritmo de preservación de dependencias, presentando en pantalla las dependencias que se han perdido en la descomposición. Finalmente, también se presenta una pantalla con las dependencias en el mismo orden en que se han ido eligiendo en cada fase del proceso de descomposición.

6. DETERMINACIÓN DE SI UNA DESCOMPOSICIÓN ES SPI

RISSANEN (1979), como hemos visto en el capítulo 5, con el concepto de proyecciones independientes, proporciona una condición fácil para comprobar si la descomposición de un esquema en dos esquemas es o no SPI, sin necesidad de apoyarse en las extensiones de los esquemas.

Sea el esquema de relación $R(AT, DEP)$ donde:

$$AT=(A,B,C)$$

$$DEP=\{AC \rightarrow B, B \rightarrow C\}$$

y sea la descomposición $R1, R2$ con

$$R1=(\{A,B\} \{ \})$$

$$R2=(\{B,C\}, \{B \rightarrow C\})$$

La descomposición es SPI, ya que cumple la condición de Rissanen, y el atributo común B es clave en $R2$.

En el caso de que el número de esquemas sea mayor de 2, se podría pensar en aplicar las condiciones por parejas. El problema principal que plantea esta opción es el de determinar el orden en que deben ser tomados los distintos esquemas para aplicarles el operador de combinación. Además, puede ocurrir que la descomposición final sea SPI y que el resultado del operador aplicado por parejas a los esquemas no lo sea.

6.1. Algoritmo de Ullman

Con carácter general, para cualquier número de esquemas, ULLMAN (1982), propone un algoritmo para determinar si una descomposición es o no SPI. El algoritmo es el siguiente:

ALGORITMO SPI_1

Entrada:

Esquema de relación $R=(AT,DEP)$

Una descomposición $PROY=R_1..R_n$ del esquema R

Salida:

SI : La descomposición $PROY$ es SPI.

NO : La descomposición no es SPI.

Proceso:

- 1) $NCOL$ = número de atributos de R .
- 2) $NFIL$ = número de esquemas de la descomposición $PROY$.
- 3) Construir una matriz MAT de $NFIL$ filas y $NCOL$ columnas, la columna j corresponde al atributo A_j de AT , y la fila i corresponde al esquema R_i de $PROY$. Los valores de las columnas, para cada una de las filas R_i de la matriz, se rellenan de la siguiente forma:

Si A_j esta en R_i , $MAT_{ij} = a_j$

Si A_j no esta en R_i , $MAT_{ij} = b_{ij}$

- 4) Mientras MAT cambie y no haya una fila de MAT en la que todos los valores son 'a':

Para toda dependencia $X \rightarrow Y \in DEP$:

- Seleccionar las filas de la matriz tales que $MAT_{icol}=MAT_{jcol}$ para todo col =número columna del atributo X_k de X .
- Para las filas seleccionadas anteriormente, transformar los valores de la columna Y de la matriz de la forma:

Si hay algún a_i entre estos valores, poner todos los valores a a_i .

Si todos los valores de la columna de la matriz son de la forma b_{ij} , b_{kj} ... poner todos los valores a un b_{ij} cualquiera.

Fin para toda dependencia $X \rightarrow Y \in \text{DEP}$

Fin_mientras.

- 5) Si se encuentra alguna fila de la matriz en la que todos los valores son 'a' SALIDA=SI, en caso contrario SALIDA=NO.

Veamos el funcionamiento del algoritmo con el siguiente ejemplo: Sea $R(AT, \text{DEP})$ el esquema de partida y $\text{PROY} = R_1, R_2, R_3, R_4, R_5$ la descomposición resultante, donde:

$AT = \{A, B, C, D, E\}$

$\text{DEP} = \{ C \rightarrow D \quad A \rightarrow C \quad B \rightarrow C \quad CE \rightarrow A \quad DE \rightarrow C \}$

$R_1 = \{A, D\} \quad R_2 = \{A, B\} \quad R_3 = \{B, E\} \quad R_4 = \{C, D, E\} \quad R_5 = \{A, E\}$

La matriz inicial de partida es:

	A	B	C	D	E
AD	a1	b12	b13	a4	b15
AB	a1	a2	b23	b24	b25
BE	b31	a2	b33	b34	a5
CDE	b41	b42	a3	a4	a5
A	Ea1	b52	b53	b54	a5

Aplicando las dependencias a la matriz, en el orden $C \rightarrow D$, $A \rightarrow C$, $B \rightarrow C$, $CE \rightarrow A$ y $DE \rightarrow C$ se obtiene finalmente la matriz:

	A	B	C	D	E
AD	a1	b12	b23	a4	b15
AB	a1	a2	b23	a4	b25
BE	a1	a2	a3	a4	a5
CDE	a1	b42	a3	a4	a5
AE	a1	b52	a3	a4	a5

Al existir encuentra una fila de la matriz (BE) donde todos los valores son "a", la descomposición es SPI.

6.2. Versión mejorada del Algoritmo de Ullman

En el algoritmo SPI_1 no hay ninguna restricción sobre el orden en el que se deben elegir las dependencias con las que se va a transformar la matriz.

En la práctica, sobre todo cuando existen atributos que son implicantes en unas dependencias e implicados en otras, observamos que el orden en que se toman las dependencias reduce o amplía el número de iteraciones que se ejecutan hasta decidir si la descomposición preserva la información.

Veámoslo con un sencillo ejemplo:

Esquema de partida $R = (\{A, B, C, D\} \{A \rightarrow B, B \rightarrow C, C \rightarrow D\})$

Descomposición:

$R_1 = (\{A, B\} \{A \rightarrow B\})$

$R_2 = (\{B, C\} \{B \rightarrow C\})$

$R_3 = (\{C, D\} \{C \rightarrow D\})$

La matriz de la que se parte es:

	A	B	C	D
AB	a1	a2	b13	b14
BC	b21	a2	a3	b24
CD	b31	b32	a3	a4

• Orden de las dependencias $C \rightarrow D$ $B \rightarrow C$ $A \rightarrow B$

Con este orden, las transformaciones sobre la matriz son las siguientes:

Con la dependencia $C \rightarrow D$

	A	B	C	D
AB	a1	a2	b13	b14
BC	b21	a2	a3	a4
CD	b31	b32	a3	a4

Con la dependencia $B \rightarrow C$

	A	B	C	D
AB	a1	a2	a3	b14
BC	b21	a2	a3	a4
CD	b31	b32	a3	a4

Con la dependencia $A \rightarrow B$

	A	B	C	D
AB	a1	a2	a3	b14
BC	b21	a2	a3	a4
CD	b31	b32	a3	a4

Con la dependencia $C \rightarrow D$

	A	B	C	D
AB	a1	a2	a3	a4
BC	b21	a2	a3	a4
CD	b31	b32	a3	a4

Como ya se encuentra una fila de la matriz cuyos valores son todos 'a' el algoritmo concluye. La descomposición es SPI, y se han necesitado 4 iteraciones.

- **Orden de las dependencias $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$**

Con este orden, las transformaciones sobre la matriz son las siguientes:

Con la dependencia $A \rightarrow B$

	A	B	C	D
AB	a1	a2	b13	b14
BC	b21	a2	a3	b24
CD	b31	b32	a3	a4

Con la dependencia $B \rightarrow C$

	A	B	C	D
AB	a1	a2	a3	b14
BC	b21	a2	a3	b24
CD	b31	b32	a3	a4

Con la dependencia $C \rightarrow D$

	A	B	C	D
AB	a1	a2	a3	a4
BC	b21	a2	a3	a4
CD	b31	b32	a3	a4

Como ya se encuentra una fila de la matriz cuyos valores son todos 'a' el algoritmo concluye. La descomposición es SPI, y se han necesitado 3 iteraciones.

Como hemos visto con este sencillo ejemplo con sólo 3 dependencias, el orden en que éstas se eligen reduce o aumenta en uno el número de iteraciones para llegar a determinar si la descomposición es o no SPI.

Con un conjunto de dependencias mucho más elevado, el ahorro o incremento de iteraciones para llegar al resultado final puede ser considerable. Por esta razón, se propone un algoritmo que ordena las dependencias que van a ser aplicadas a la matriz, y que permite acelerar el proceso para determinar si una descomposición es o no SPI.

En el algoritmo que determina si una descomposición es SPI, el aplicar la dependencia $X \rightarrow Y$ a una matriz, consiste en transformar la columna Y de la matriz en función de los valores que se encuentren en la columna X de la matriz.

Supongamos que tenemos sólo dos dependencias de la forma $X \rightarrow Y$, $Y \rightarrow Z$. Como podemos observar, el atributo Y es a la vez implicante en una dependencia e implicado en otra.

Sean:

numx=número de filas de la matriz cuyos valores coinciden en los atributos de la columna X.

numy=número de filas de la matriz cuyos valores coinciden en los atributos de la columna Y.

La aplicación de la dependencia $X \rightarrow Y$ transformará los valores de la columna del atributo Y. Sea numy' el número de filas de la matriz cuyos valores coinciden en los atributos de Y. En el caso peor (no ha habido ningún cambio) $\text{numy}' = \text{numy}$. Pero si se ha producido algún cambio, $\text{numy}' > \text{numy}$, con lo que al aplicar la dependencia $Y \rightarrow Z$ a la matriz, el número de filas de las que se parte para producir la transformación en la matriz (numy') es mayor y por lo tanto se pueden transformar más valores en la columna del atributo Z que si el orden en el que se hubieran aplicado las dependencias a la matriz hubiera sido el inverso ($Y \rightarrow Z, X \rightarrow Y$).

Por lo tanto, en el caso de que tengamos dependencias con atributos que son implicantes en unas e implicados en otras, siempre se aplicarán a la matriz las dependencias en las que estos atributos sean implicados y, posteriormente, las dependencias en las que estos atributos sean implicantes.

En el caso de que tengamos dependencias en los que los descriptores implicantes tienen más de un atributo, empezaremos por las dependencias que sólo tienen un atributo como implicante, después las que tienen dos y así sucesivamente, ya que cuanto mayor sea el número de atributos del implicante, menor será el número de filas de las que partamos para aplicar las transformaciones a la matriz, y el objetivo que persigue este algoritmo es justamente el contrario.

El algoritmo que a continuación proponemos LÓPEZ (1988), analiza las dependencias determinando si los atributos que las componen son sólo implicantes, implicados o ambas cosas, y asigna un valor a cada una de estas dependencias. La dependencia elegida será la que tenga más atributos que sean sólo implicantes, puesto que desencadenarán mayores transformaciones en la matriz. El algoritmo es el siguiente:

ALGORITMO ORDENA

Sea $|X|$ el número de atributos del descriptor X.

Entrada:

Un conjunto DEP de dependencias funcionales

Salida:

Un conjunto ORDEN de dependencias funcionales.

Proceso:

- 1) Agrupar el conjunto DEP en subconjuntos CONJUNTO_i , de forma que:

$$\text{CONJUNTO}_i = \{X \rightarrow Y \in \text{DEP} \text{ tal que } |X|=i\}$$

2) Para todo CONJUNTO_i, $i=1..n$ hacer:

Para cada dependencia $X \rightarrow Y \in \text{CONJUNTO}_i$, calcular

VALOR($X \rightarrow Y$) de la siguiente forma:

VAL=0

CONJUNTOXY= Atributos de X U Y.

Para cada AT $\in \text{CONJUNTOXY}$ hacer:

- Si AT aparece en CONJUNTO_i siempre como implicante:
VAL=VAL+0
- Si AT aparece en CONJUNTO_i como implicante e implicado:
VAL=VAL+1
- Si AT aparece en CONJUNTO_i siempre como implicado:
VAL=VAL+2

Fin_para_cada AT

VALOR(DEP)=VAL

Fin_para_cada_dependencia

ORDEN=ORDEN U $\{X \rightarrow Y\} \mid \text{VALOR}(X \rightarrow Y) \min (\text{VALOR}(Y \rightarrow Z))$
para toda $Y \rightarrow Z \in \text{CONJUNTO}_i$.

Fin para todo CONJUNTO_i.

Veamos cómo trabajaría el algoritmo con el conjunto de dependencias:

$\{A \rightarrow C \quad C \rightarrow D \quad B \rightarrow C \quad CE \rightarrow A \quad DE \rightarrow C\}$

En el paso 1 los conjuntos que se forman son:

CONJUNTO₁= $\{A \rightarrow C \quad C \rightarrow D \quad B \rightarrow C\}$

CONJUNTO₂= $\{CE \rightarrow A \quad DE \rightarrow C\}$

En el paso 2 del algoritmo, los valores que se asignan a cada una de las dependencias

son:

CONJUNTO1:

Valor ($A \rightarrow C$)= 1

Valor ($C \rightarrow D$)= 3

Valor ($B \rightarrow C$)= 1

CONJUNTO2:

Valor ($C, E \rightarrow A$)=3

Valor ($D, E \rightarrow C$)=1

El orden de las dependencias que determina el algoritmo es:

{ $A \rightarrow C$ $B \rightarrow C$ $C \rightarrow D$ $DE \rightarrow C$ $CE \rightarrow A$ }

Para las dependencias con un solo atributo como implicante (CONJUNTO1), vemos que el atributo C es a la vez implicado e implicante, por lo tanto, la última dependencia de este grupo es aquella en la que C es implicante, es decir, $C \rightarrow D$. A las otras dos dependencias de este conjunto ($A \rightarrow C$ y $B \rightarrow C$), el algoritmo les asigna igual valor, por lo tanto, el orden entre ellas es indiferente.

Para las restantes dependencias del CONJUNTO2 ($CE \rightarrow A$, $DE \rightarrow C$), el atributo C también es implicante e implicado, por lo tanto, la primera dependencia que elige el algoritmo es $DE \rightarrow C$.

El algoritmo propuesto por Ullman ordenando previamente las dependencias (con el algoritmo ORDENA) queda como sigue:

ALGORITMO SPI_2

Entrada:

Esquema de relación $R=(AT, DEP)$

Una descomposición $PROY=R_1..R_n$ del esquema R

Salida:

SÍ: La descomposición PROY es SPI.

NO: La descomposición no es SPI.

Proceso:

- 1) NCOL = número de atributos de R
- 2) NFIL = número de esquemas de la descomposición PROY.
- 3) Construir una matriz MAT de NFIL filas y NCOL columnas, la columna j corresponde al atributo A_j de AT, y la fila i corresponde al esquema R_i de PROY. Los valores de las columnas para cada una de las filas R_i , de la matriz se rellenan de la siguiente forma:

Si A_j esta en R_i , $MAT_{ij} = a_j$

Si A_j no esta en R_i , $MAT_{ij} = b_{ij}$

- 4) Mientras MAT cambie y no hay una fila de MAT a a's:

ORDEN(DEP)= Salida algoritmo **ORDENA** de ordenación de dependencias

Para toda dependencia $X \rightarrow Y$ de ORDEN(DEP):

- Seleccionar las filas de la matriz tales que $MAT_{icol}=MAT_{jcol}$ para todo col =número columna del atributo X_k de X.
- Para las filas seleccionadas anteriormente, transformar los valores de la columna Y de la matriz de la forma:

Si hay algún a_i entre estos valores, poner todos los valores a a_i .

Si todos los valores de la columna de la matriz son de la forma b_{ij} , b_{kj} ... poner todos los valores a un b_{ij} cualquiera.

Fin_para_toda_dependencia

Fin_mientras.

- 5) Si se encuentra alguna fila de la matriz en la que todos los valores son a's SALIDA=SI, en caso contrario SALIDA=NO.

El algoritmo SPI_2 es más eficiente que el SPI_1, aunque no hay que olvidar que el algoritmo SPI_2 tiene un coste adicional que es el cálculo del orden de las dependencias antes de trabajar con la matriz (que es polinomial al número de dependencias del esquema de partida). En LÓPEZ (1998) se realiza un análisis del tiempo de respuesta de ambos algoritmos, comprobando que en prácticamente todos los casos, el tiempo que se necesita

para ordenar las dependencias y llegar al resultado final es menor que el tiempo que el algoritmo tradicional emplea en hacer todas las iteraciones extra que necesita hasta llegar al resultado final²¹.

7. DETERMINACIÓN DE SI UNA DESCOMPOSICIÓN PRESERVA LAS DEPENDENCIAS

El concepto de proyecciones independientes proporciona una condición fácil para comprobar si la descomposición de un esquema en dos preserva las dependencias. Pero esta condición no es aplicable cuando el esquema de partida se descompone en más de dos esquemas.

En teoría, es fácil determinar si una descomposición $p=R_1 \dots R_n$, preserva un conjunto de dependencias DEP. Para ello, bastaría con:

1. Calcular DEP^+
2. Para cada R_i , calcular $DEP[R_i]$
3. Hacer $G = \bigcup DEP[R_i]_{i=1..n}$
4. Si G es un recubrimiento de DEP, entonces la descomposición preserva las dependencias.

En la práctica, este proceso tal y como se ha descrito no puede ser implementado debido a su alto coste computacional, puesto que el tiempo de cálculo de DEP^+ es exponencial al número de dependencias de DEP.

Ullman (1982) propone un algoritmo de coste computacional menor en el que no es necesario calcular DEP^+ . El tiempo de cálculo es polinomial al número de dependencias de DEP. El algoritmo es el siguiente:

ALGORITMO PDEP²²

Entrada:

Un esquema de relación $R=(AT, DEP)$

Una descomposición $p=R_1 \dots R_n$ del esquema R

²¹ En la herramienta RENO existe una opción independiente de menú que incluye el algoritmo para comprobar si una descomposición es SPI, con el objetivo principal de permitir al usuario comprobar si una descomposición realizada por él de forma manual preserva o no la información.

²² En la herramienta RENO, este algoritmo aparece como una opción independiente de menú con el objetivo de permitir al usuario comprobar si una descomposición realizada por él, de forma manual, preserva o no las dependencias funcionales del esquema de partida. También se aplica este algoritmo en el paso final del algoritmo de descomposición a esquemas en FNBC a fin de indicar si en la descomposición resultante se han perdido dependencias y, si esto ocurre, las dependencias que no se han preservado se presentan en pantalla.

Salida:

Una decisión sobre si p preserva o no las dependencias.

Proceso:

1) $G = \bigcup_{i=1..n} \text{DEP}[R_i]$

2) $\text{RESULTADO} = \text{SI}$

3) Para toda $X \rightarrow Y \in \text{DEP}$ hacer:

$Z = X$

Mientras Z cambie

Para $i=1..n$ hacer

$Z = Z \cup ((Z \cap R_i)^+ \cap R_i)$

Fin_para

Fin_mientras

Si $Y \not\subseteq Z$ entonces

$\text{RESULTADO} = \text{NO}$

Fin_si

Fin para toda $X \rightarrow Y \in \text{DEP}$.

El objetivo de este algoritmo es determinar si el conjunto G es un recubrimiento de DEP . Para ello, para cada dependencia $X \rightarrow Y$ de DEP , habrá que comprobar si Y está incluido en X_G^+ .

El cálculo de X_G^+ , se realiza considerando repetidamente el cierre de X respecto a las distintas proyecciones de DEP sobre cada uno de los esquemas R_i 's. Para ello, se define la R_i -operación sobre un conjunto Z de atributos con respecto a un conjunto DEP de dependencias, como la sustitución de Z por $Z \cup ((Z \cap R_i)^+_{\text{DEP}} \cap R_i)$. Esta R_i -operación añade a Z aquellos atributos A para los que se cumple que $(Z \cap R_i) \rightarrow A \subseteq \text{DEP}[R_i]$.

Así, para calcular X_G^+ se comienza con X , y para cada esquema R_i se aplica repetidamente las R_i -operación. El cálculo acabará cuando para ninguno de los esquemas R_i , la R_i -operación produzca un cambio en el valor de X_G^+ .

Veamos cómo trabajaría el algoritmo con el esquema de relación $R(AT, DEP)$ donde:

$$AT = \{A, B, C, D\}$$

$$DEP = \{A \rightarrow B \quad B \rightarrow C \quad C \rightarrow D \quad D \rightarrow A\}$$

Y sea $p=R_1, R_2, R_3$ una descomposición de R donde:

$$R_1 = (\{A, B\} \quad \{A \rightarrow B \quad B \rightarrow A\})$$

$$R_2 = (\{B, C\} \quad \{B \rightarrow C \quad C \rightarrow B\})$$

$$R_3 = (\{C, D\} \quad \{C \rightarrow D \quad D \rightarrow C\})$$

Recordemos que en el cálculo de la proyección del conjunto DEP sobre los distintos esquemas, en realidad lo que se proyecta es el conjunto DEP^+ , de ahí que, por ejemplo, en la proyección sobre el esquema AB , no sólo aparece la dependencia $A \rightarrow B$ (que originalmente estaba en el conjunto DEP de partida) sino que también aparecerá la dependencia $B \rightarrow A$.

En principio, parece que la dependencia $D \rightarrow A$ es la única que se ha perdido en el proceso de descomposición. Aplicando el algoritmo para dicha dependencia, obtenemos los siguientes resultados:

$$1. \quad G = DEP[AB] \cup DEP[BC] \cup DEP[CD]$$

$$2. \quad Z = \{D\}$$

3. Aplicando la R -operaciones sobre Z :

$$AB\text{-operación } Z = \{D\}$$

$$AB\text{-operación } Z = \{A, B, C, D\}$$

$$BC\text{-operación } Z = \{D\}$$

$$BC\text{-operación } Z = \{A, B, C, D\}$$

$$CD\text{-operación } Z = \{C, D\}$$

$$CD\text{-operación } Z = \{A, B, C, D\}$$

$$AB\text{-operación } Z = \{C, D\}$$

$$AB\text{-operación } Z = \{A, B, C, D\}$$

$$BC\text{-operación } Z = \{B, C, D\}$$

$$BC\text{-operación } Z = \{A, B, C, D\}$$

$$CD\text{-operación } Z = \{B, C, D\}$$

$$CD\text{-operación } Z = \{A, B, C, D\}$$

4. Como $D_G^+ = \{A, B, C, D\}$ la dependencia $D \rightarrow A$, se preserva en la descomposición.

8. ALGORITMO GRÁFICO PARA EL PARTICIONAMIENTO VERTICAL²³

Como ya hemos visto, el particionamiento vertical es una de las formas de reorganizar datos a fin de aumentar la eficiencia. Los algoritmos para el particionamiento tienen como objetivo crear grupos de atributos de una tabla que sean accedidos en conjunto por un gran número de transacciones.

Este algoritmo representa una considerable mejora respecto a otros propuestos hasta la fecha, ya que no es necesario un particionamiento binario iterativo, no requiere una función objetivo y, además, su complejidad es menor que la de los anteriores algoritmos de particionamiento vertical.

Aquí sólo expondremos, de forma resumida, unas ideas sobre este algoritmo y lo aplicaremos a un ejemplo; remitimos al lector interesado en este tema a NAVATHE (1989).

8.1. Conceptos básicos

A) Matriz y grafo de afinidad

El algoritmo comienza con la construcción de la *matriz de utilización de atributos*, que representa el uso que las transacciones representativas hacen de los atributos de la relación. Como se puede observar en la figura 7.2, cada fila se refiere a una transacción, mientras que en las columnas se colocan los atributos de la relación, poniendo un 1 si una transacción utiliza el atributo y 0 en caso contrario.

A partir de esta matriz se genera la denominada *matriz de afinidad de atributos*, definiéndose:

$$\text{aff}_{ij} = \sum_{k \in r} \text{acc}_{ij}^k$$

donde acc es el número de accesos por unidad de tiempo de la transacción k que referencia simultáneamente a los atributos i y j . El sumatorio se calcula sobre todas las transacciones que pertenecen al conjunto de transacciones que pertenecen al conjunto de transacciones significativas.

²³ Aunque este algoritmo ha sido programado no se ha incluido en RENO por no haberse probado suficientemente.

MATRIZ DE UTILIZACION DE ATRIBUTOS											Número de accesos por unidad de tiempo
T. A	CODIGO	TITULO	IDIOMA	TEMA	NUM.	AÑO	EDIT.	PRECIO	EDITOR	OBSER.	
T1	0	1	1	0	1	1	1	1	0	1	ACC1=5
T2	1	1	0	0	0	0	0	0	0	0	ACC2=10
T3	1	1	1	1	0	1	1	0	1	0	ACC3=15
T4	1	1	0	0	1	1	1	0	1	1	ACC4=10
T5	1	1	0	0	0	1	1	0	0	1	ACC5=5

Figura 7.2. Matriz de utilización de atributos

La afinidad de los atributos mide la fuerza del enlace entre atributos en una misma transacción. En la figura 7.3 puede observarse la matriz de afinidad de atributos derivada de la matriz de utilización anterior. Téngase en cuenta que el valor de los elementos de la diagonal es igual a la suma de los elementos de la columna donde aparece el atributo, representando de esta manera la fuerza que posee un atributo debida a su utilización por el conjunto de las transacciones.

	CODIGO	TITULO	IDIOMA	TEMA	NUM.	AÑO	EDIT.	PRECIO	EDITOR	OBSER.
CODIGO	40	40	15	15	10	30	30	0	25	15
TITULO	40	45	20	15	15	35	35	5	25	20
IDIOMA	15	20	20	15	5	20	20	5	15	5
TEMA	15	15	15	15	0	15	15	0	15	0
NUM.	10	15	5	0	15	15	15	5	10	15
AÑO	30	35	20	15	15	35	35	5	25	20
EDIT.	30	35	20	15	15	35	35	5	25	20
PRECIO	0	5	5	0	5	5	5	5	0	5
EDITOR	25	25	15	15	10	25	25	0	25	10
OBSER.	15	20	5	0	15	20	20	5	10	20

Figura 7.3. Matriz de afinidad de atributos

A partir de la matriz de afinidad de atributos se puede construir de forma inmediata el grato de afinidad, en el cual los valores de los arcos representan la afinidad entre dos atributos (véase figura 7.4).

Basándose en este grafo se construye un árbol de extensión²⁴ conectado linealmente (esto es, que tiene únicamente dos extremos) sobre el que se van formando ciclos que darán lugar a los fragmentos de la relación.

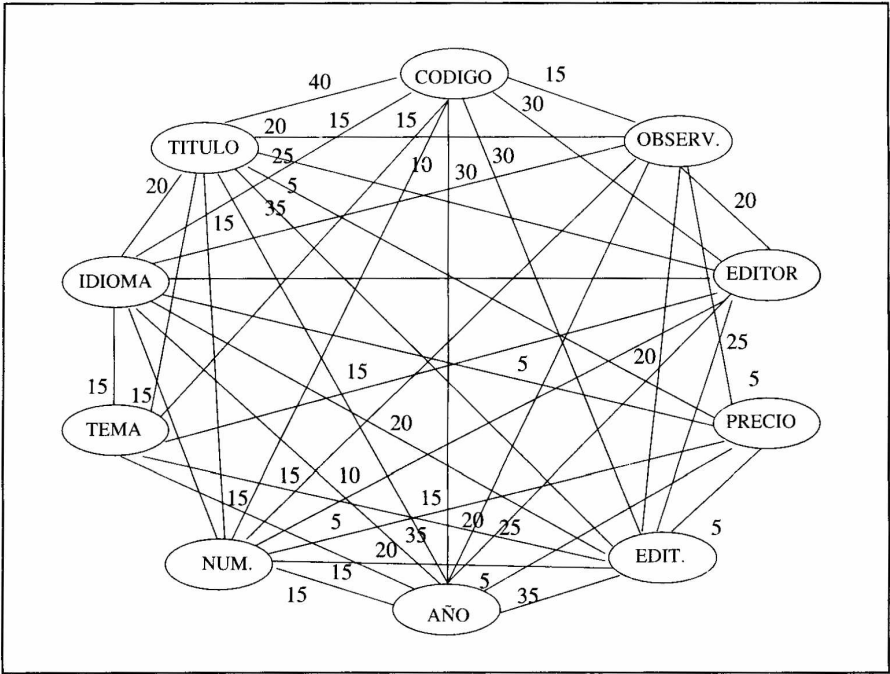


Figura 7.4. Grafo de afinidad

B) Ciclo de afinidad y extensión de un ciclo

Denominaremos **ciclo primitivo** a cualquier ciclo del grafo de afinidad, *nodo ciclo* al nodo del arco que completa un ciclo que fue elegido en primer lugar y *ciclo de afinidad* al ciclo primitivo que contiene un nodo ciclo.

Llamaremos *arco anterior* a un arco que haya sido seleccionado entre el último corte y el nodo ciclo. Se denomina *extensión de un ciclo* al ciclo que se extiende pivotando en el nodo ciclo.

²⁴ Un árbol de extensión es un subgrafo de un grafo conectado G. Este subgrafo es un árbol e incluye todos los nodos de G.

A1 Explicaremos a continuación el mecanismo de formación de ciclos (que, como ya hemos dicho, darán lugar a los fragmentos verticales); supongamos que, como se muestra en la figura 7.5, ya han sido seleccionados los arcos *a* y *b*, y el próximo a seleccionar resulta el *c*. Este arco, junto con *a* y *b*, forma un ciclo primitivo; para determinar si forma un ciclo de afinidad, hay que comprobar la llamada *condición de existencia de un ciclo de afinidad*, esto es:

- que no exista un arco anterior, o
- que el peso del arco anterior sea menor o igual que el peso de todos los arcos del ciclo

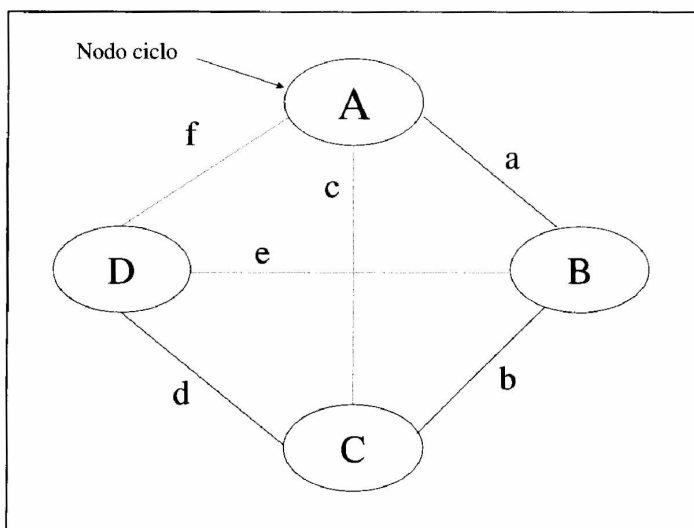


Figura 7.5. Ciclo y extensión de un ciclo

En el caso de la figura, *a*, *b*, *e* es un ciclo de afinidad al no haber un arco anterior; entonces se marca como partición candidata y el nodo A (al ser el primero seleccionado de los dos nodos que tiene el arco *e*) se convierte en el nodo ciclo.

Veamos a continuación cómo se lleva a cabo la extensión de un ciclo. Supongamos que, una vez determinado el ciclo *a*, *b*, *e*, se selecciona el arco *d* (véase figura 7.6), habrá que comprobar si se puede realizar la extensión, para ello la condición de posibilidad de extensión es que el peso del arco considerado o del arco que completa sea mayor o igual que cualquiera de los arcos que forman el ciclo, por tanto, el ciclo *a*, *b*, *c* se extenderá al ciclo *a*, *b*, *c*, *d*, *f* si el peso del arco considerado *d* o del arco que completa el ciclo *f* es mayor o igual al minimal del peso de los arcos *a*, *b* y *c*.

Existen dos formas de crear una partición:

- Con un arco nuevo: en el ejemplo anterior el arco *d* no había sido considerado anteriormente, por lo que se considera *arco nuevo*. Si *d* y *f* cumplen la condición de posibilidad de extensión, entonces el nuevo ciclo es *a, b, d, f* pero si no la cumplen se produce un corte sobre el arco *d* aislando al ciclo *a, b, e*, que se considera como una partición.

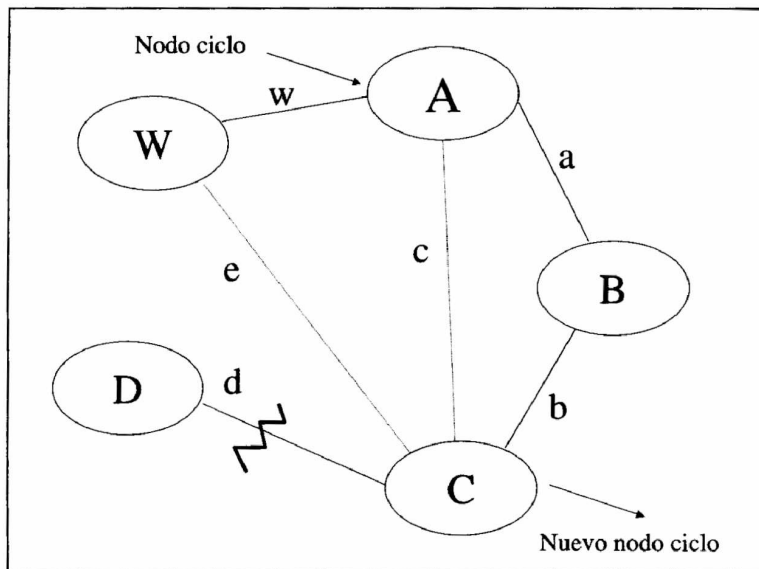


Figura 7.6. Partición

- Con un arco anterior: si al cortar un arco nuevo existe un arco anterior, se cambia el nodo ciclo, que pasa a ser el nodo incidente del arco cortado y se estudia la posibilidad de extensión del ciclo con el arco anterior. En la figura 7.6, si *a, b, c* forman un ciclo siendo A el nodo ciclo, y se produce un corte en *d*, existiendo un arco anterior *w*, entonces C se convierte en el nuevo nodo ciclo, ya que el arco cortado *d* se origina en C. Ahora se evalúa la posibilidad de extender el ciclo *a, b, e* al ciclo *a, b, e, w*. Si *w* o *e* no satisfacen la posibilidad de extensión entonces se produce un corte en *w*; C sigue siendo el nodo ciclo y *a, b, c* se convierte en una partición.

Si, por el contrario, la condición de extensión es satisfecha, se extiende el ciclo a *w, a, b, e*, permaneciendo C como el nodo ciclo y no se puede formar todavía ninguna partición.

ALGORITMO PI (particionamiento vertical)**Entrada:**

A_j , conjunto de atributos.

Salida:

$R(T_i, D_i)$ conjunto de esquemas **particionados**.

Proceso:

- 1) Construir el grafo de afinidad de atributos de la relación considerada.
- 2) Empezar por cualquier nodo.
- 3) Seleccionar un arco que satisfaga las siguientes condiciones:
 - que esté linealmente conectado al árbol construido hasta el momento,
 - que posea el mayor valor de afinidad de los arcos de los extremos del árbol.

Esta iteración finaliza cuando se seleccionan todos los nodos.

- 4) Cuando el siguiente arco forma un ciclo:
 - Si el nodo ciclo no existe, verificar la posibilidad de un ciclo y, si se cumple, marcar el ciclo como un ciclo de afinidad y considerarlo como una partición candidata. Ir al paso 3.
 - Si el nodo ciclo ya existe, descartar este arco e ir al paso 3.
- 5) Cuando el siguiente arco seleccionado no forma un ciclo y existe una partición candidata:
 - Si no existe un arco anterior, verificar la posibilidad de extensión del ciclo. Si no hay posibilidad, cortar el arco y considerar el ciclo como una partición.
 - Si existe un arco anterior, cambiar el nodo ciclo y verificar la posibilidad de extensión del ciclo con el arco anterior. Si no hay posibilidad, cortar el arco anterior y considerar el ciclo como una partición. Ir al paso 3.

Ejemplo

Consideramos la relación:

LIBRO (*código, título, idioma, lema, número, año, editorial, precio, editor, observaciones*)

en la cual suponemos que se dan las afinidades entre los atributos que aparecen en la matriz de la figura 7.2.

Empecemos a aplicar el algoritmo por el nodo *código* (el algoritmo produce el mismo particionamiento, independientemente del nodo del que se parta) seleccionando el arco de *código* a *título*, ya que presenta el valor de mayor afinidad. A continuación tenemos que buscar el arco de mayor valor que salga de *código* o *título*, y elegimos *año* que tiene un valor 35 (también podríamos haber escogido el que sale de *título* y llega a *editorial*). Ahora debemos seleccionar el arco de mayor valor que parta de *código* o de *año*, que resulta ser el de *año* y *editorial* de valor 35.

Teniendo en cuenta las mismas condiciones, el próximo arco a seleccionar es el existente entre *código* y *editorial*, de valor 30; como forma un ciclo primitivo, entramos en el paso 3, al no existir un nodo ciclo, debemos verificar la posibilidad de un ciclo de afinidad, que se cumple ya que no existe ningún arco anterior por lo que, en definitiva, *código, título, año, editorial* forman una partición candidata y *código* se convierte en un nodo ciclo, véase figura 7.7.

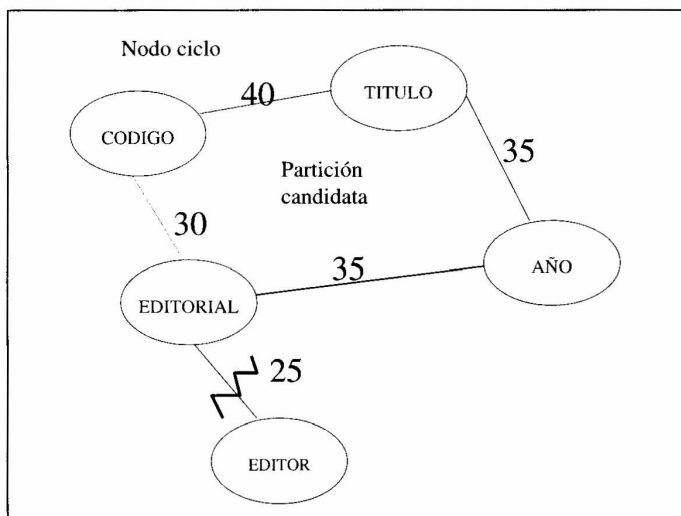


Figura 7.7. Aplicación del algoritmo (I)

A continuación debemos seleccionar un arco que partiendo de *código* o de *editorial* posea el mayor valor de afinidad, por ejemplo el que une *editorial* con *editor*, de valor 25. Al no formar este arco un ciclo y debido a que existe una partición candidata, entramos en el paso 5. Como no existe un arco anterior, hay que comprobar la condición de extensión del ciclo con este nuevo arco, condición que no se cumple ya que el arco considerado (*editorial-editor*) y el que completa el ciclo (*código-editor*) poseen de valor 25, que es menor que cualquiera de los arcos que toman la partición candidata; por tanto, se debe cortar por este arco y considerar el ciclo como partición, volviendo al paso 3, véase figura 7.8.

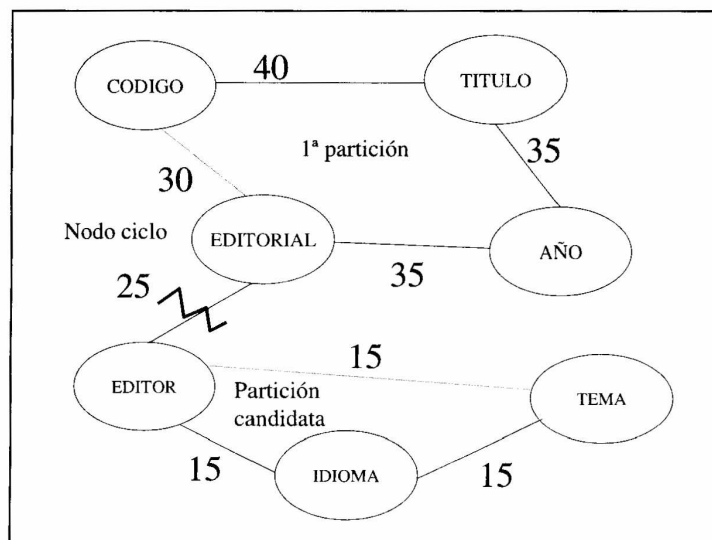


Figura 7.8. Aplicación del algoritmo (II)

El próximo arco seleccionado es el existente entre, por ejemplo, *editor* e *idioma*, de valor 15, posteriormente podemos seleccionar el que va de *idioma* a *tema*, también de valor 15, y el de *editor* a *tema*, que vendría a formar un ciclo primitivo, con lo que entramos en el paso 4, comprobamos la posibilidad de formar un ciclo de afinidad (que se cumple al no haber arcos anteriores), por lo que *editor*, *idioma* y *tema* se marcan como ciclo de afinidad y se considera partición candidata.

Seguidamente tenemos que escoger el arco que va de *código* a *observaciones*, de valor 15, con lo que entramos en el paso 5, y, al no existir un arco anterior, verificamos la condición de extensión del ciclo que no se cumple, ya que *tema*, *idioma*, *observación* no incluye el nodo ciclo, por lo que no se verifica la extensión y se produce un corte en el arco de *código* a *observaciones* y se considera el ciclo *editor*, *idioma*, *tema* como otra partición.

Por último, se escoge el arco de *observaciones a número*, de valor 15, y de éste a *precio*, formándose así la última partición, véase figura 7.9.

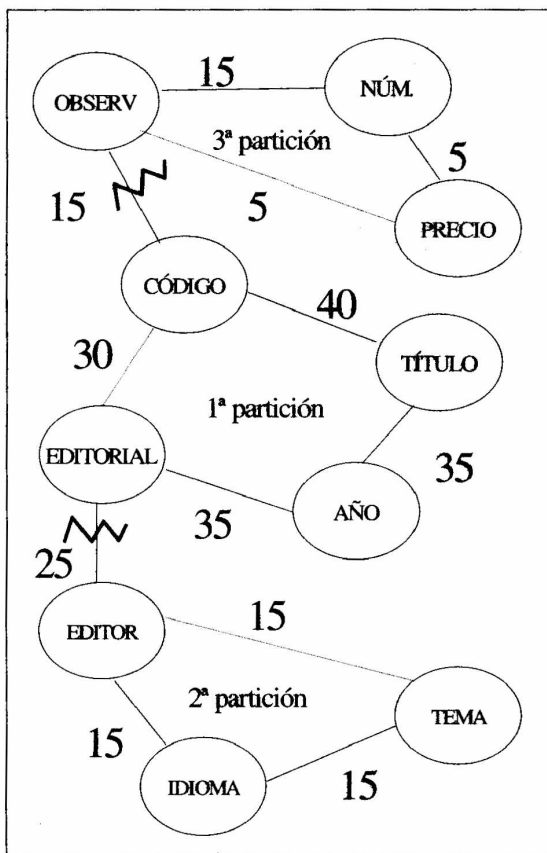


Figura 7.9. Aplicación del algoritmo (III)

En definitiva, habíamos partido de la relación:

LIBRO (*código, título, idioma, tema, número, año, editorial, precio, editor, observaciones*)

y aplicando este algoritmo hemos llegado a formar tres relaciones (más reducidas) que se reparten los atributos de la original en función de su utilización:

LIBRO1 (*código, título, editorial, año*)

LIBRO2 (*código, editor, idioma, tema*)

LIBRO3 (*código, número, precio, observación*)

lo que nos proporciona una serie de ventajas: reduce el número de E/S, mejora la concurrencia, etc.; aunque se penalizan las consultas resultantes de la combinación de estas tablas, además de introducirse las correspondientes restricciones de integridad referencial.

Se podría llegar a una partición intermedia de la relación original y considerando la instrumentación de dos relaciones en lugar de las tres propuestas en el algoritmo:

LIBRO1 (*código, título, editorial, año*)

LIBRO2 (*código, editor, tema, idioma, observaciones, número, precio*)

9. ALGUNAS CONSIDERACIONES RELATIVAS A LA EFICIENCIA Y CONCLUSIONES

Hemos detallado en este capítulo diversos algoritmos de diseño de bases de datos relacionales (proceso de normalización y particionamiento vertical). Para alguno de estos algoritmos, que se pueden encontrar descritos en libros y artículos, existen propuestas que los mejoran sustancialmente, en especial, en lo relativo a la eficiencia; en otros casos, hemos presentado nuestras propias propuestas de mejora, tanto relativas a la funcionalidad como a la eficiencia.

En la herramienta RENO se han implementado estos algoritmos (a excepción del de particionamiento vertical), lo que nos ha permitido evaluar su eficiencia y hacer comparaciones. Con esta finalidad se ha creado un gran número de conjuntos de dependencias, ejecutándose los distintos algoritmos para cada uno de ellos; algunos de estos conjuntos pueden representar esquemas de bases de datos reales, mientras que otros constituyen casos atípicos que podrían considerarse como ejemplos de laboratorio.

Los resultados obtenidos (PINILLA, 1992; LÓPEZ, 1998), demuestran que se obtienen ganancias de eficiencia significativas en el recubrimiento mínimo, a excepción de cuando se trata de conjuntos de dependencias atípicos. En el algoritmo de síntesis de Bernstein, partiendo ya de un recubrimiento mínimo, la mejora de eficiencia es más significativa todavía, superando por término medio el 70%. Por último, para el algoritmo de determinación de claves, el incremento de eficiencia supera, en media, el 90%, a excepción de un caso en que ambos algoritmos dan resultados similares. En lo que se refiere a los resultados de los algoritmos de síntesis y de análisis, la eficiencia es algo menor únicamente en aquellos casos en los que mejora la funcionalidad.

En general, el mejor o peor comportamiento de todos los algoritmos es función de las características del conjunto de dependencias del esquema de partida, siendo muy significativa en aquellos casos en los que se han eliminado cálculos redundantes.

De todos modos, es importante tener en cuenta que los procesos algorítmicos de normalización de bases de datos relacionales son útiles en el sentido de que pueden obtenerse de forma automática esquemas normalizados correctos sin necesidad de la pericia del diseñador, como ocurre en un diseño tradicional; sin embargo, su principal inconveniente es que no tienen en cuenta la semántica del esquema de partida, por lo que en ocasiones llegan a resultados correctos teóricamente, pero absurdos semánticamente. Esto se debe principalmente a que todas las transformaciones de los datos que realizan los algoritmos son de carácter sintáctico. A pesar de esta objeción, los algoritmos de normalización son especialmente útiles en rediseños o ampliaciones de las bases de datos relacionales, donde en ocasiones es bastante costoso comprobar manualmente el impacto de la inclusión o eliminación de una o varias dependencias.

Respecto a las nuevas propuestas algorítmicas que se han descrito anteriormente, cabe decir que su comportamiento en casos prácticos concuerda con el esperado teóricamente. El aumento de la eficiencia en casos reales es lo suficientemente significativo como para hacer posible que se incluyan en herramientas de diseño. En contraste con las mejoras que se obtienen en casos reales, se encuentra la penalización que a veces sufren los esquemas experimentales o de laboratorio.

El prototipo de herramienta desarrollado (RENO) cubre dos objetivos principalmente. El primero y más importante, de carácter didáctico, ya que permite realizar un análisis comparativo del comportamiento tanto de los nuevos algoritmos como de los tradicionales. El segundo, de carácter práctico, está enfocado a su utilización para la realización de diseños reales.

KVS@C

8. Proceso de creación y metodología de desarrollo de bases de datos
9. Modelado conceptual
10. Diseño lógico estándar
11. Diseño lógico específico y diseño físico

*P
A
R
T
E*

III

PROCESO DE CREACIÓN Y METODOLOGÍA DE DESARROLLO DE BASES DE DATOS

El desarrollo de una base de datos se integra dentro de un proceso más amplio que alcanza desde que se toma la iniciativa de crear la base de datos hasta que se encuentra totalmente operativa. En este capítulo se resume el ciclo de vida de una base de datos, abordando tanto sus aspectos técnicos como organizativos. Por último se presenta una metodología para el desarrollo de bases de datos relacionales, estudiando sus elementos y sus características.

1. INTRODUCCIÓN AL CICLO DE VIDA DE UNA BASE DE DATOS

La creación de una base de datos es, generalmente, una operación difícil, larga y costosa, que no puede improvisarse. No se trata solamente de un problema técnico, ya que las repercusiones que esta decisión puede tener en todos los niveles de la empresa (transferencia de responsabilidades entre personas y servicios, reorganización del departamento de informática, formación de los usuarios, cambio de determinados métodos de trabajo, etc.) hacen de ella una decisión que atañe a la política empresarial, por lo que no debe ser abordada en exclusiva por los técnicos.

Así pues, la responsabilidad de las decisiones relativas a todo el proceso de creación de una base de datos no corresponde únicamente a los informáticos, sino que, en ciertas fases, son los directivos y los usuarios quienes tienen, incluso, el protagonismo.

A continuación vamos a exponer de forma sistemática las distintas fases que comprende la puesta en marcha de un sistema de información orientado hacia las bases de datos, las cuales se encuentran resumidas en la figura 8.1¹.

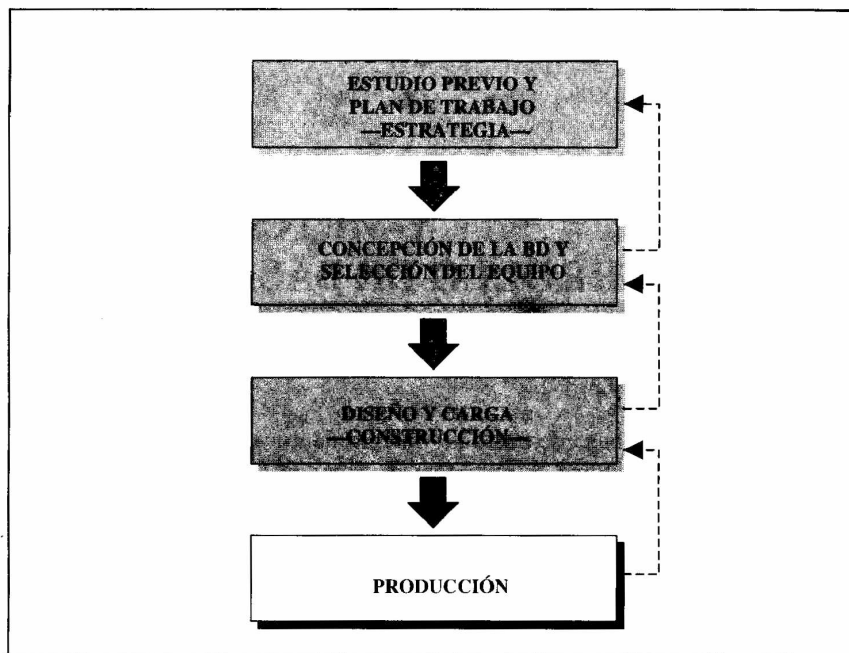


Figura 8.1. Resumen de las fases para la puesta en marcha de una BD

2. ESTUDIO PREVIO Y PLAN DE TRABAJO

2.1. Decisión política y fijación de objetivos (estudio de viabilidad)

Esta fase, a veces llamada de análisis previo, *estudio de oportunidad* o *viabilidad*, debe preceder a cualquier operación de concepción o diseño de una base de datos; en ella se ha de concretar la voluntad de los directivos de abordar el proyecto, definiendo unos objetivos claros y concretos que sirvan de pauta en todo el desarrollo.

El estudio de viabilidad, de muy corta duración y en el que la intervención de los técnicos es limitada, es fundamental para conseguir que el nuevo sistema de información, articulado alrededor de la base de datos, una vez puesto en marcha se

¹ Son las fases que aparecen en las metodologías de desarrollo de sistemas de información, pero aquí nos hemos centrado en los datos, mientras que, en general, las metodologías de desarrollo están más enfocadas hacia las funciones.

integre en el organismo, adaptándose a sus objetivos y prestando el servicio que de él se esperaba cuando se decidió su implantación.

Los técnicos implicados en el proceso de creación de un sistema de base de datos tienen que tener presente que, si no cuentan con el pleno apoyo de los directivos (los cuales no sólo han de conocer que se va a abordar el proyecto, sino que también han de comprender el alcance y significado del mismo), será mejor abandonar la idea por el momento, aplazándola para otra ocasión más propicia, ya que si no existe una decidida voluntad de la organización en su conjunto, impulsada por sus directivos, de llevar a buen fin el sistema, aumentan las probabilidades de fracaso, por lo que los costes del proyecto, a menudo muy elevados, no estarían justificados.

2.2. Evaluación previa de medios y costes

Una vez que la dirección ha tomado la decisión inicial de emprender las operaciones que conduzcan al establecimiento de un sistema de base de datos y ha definido los objetivos generales del proyecto, es preciso realizar una evaluación aproximada de los medios y de los costes que requerirá la puesta en marcha del sistema.

Se tratará sólo de dar un orden de magnitud del coste global del proyecto, ya que será prácticamente imposible, sin un estudio más profundo del sistema que se va a desarrollar, hacer un análisis detallado de costes. Sin embargo, es imprescindible que a los responsables del organismo se les proporcione una cifra aproximada de los gastos que representará la implantación del sistema y su mantenimiento, así como de los medios, en especial del personal, que van a ser precisos.

2.3. Aprobación de una estructura orgánica

Antes de comenzar el desarrollo del sistema será preciso definir la organización de la unidad administrativa que tendrá la responsabilidad de la gestión y control de la base de datos, así como determinar la estructura y los componentes del equipo encargado del desarrollo.

Las funciones del administrador de la base de datos, su responsabilidad respecto al contenido de la base, la actualización de la misma, la estandarización de la información, etc., son aspectos fundamentales que han de ser considerados desde un principio y que pueden ser decisivos para conseguir que el proyecto llegue a buen fin.

No es posible establecer una normativa general que determine cuál es la mejor organización o que lleve a definir de forma óptima las funciones y responsabilidades

del administrador de la base, porque, en cada caso, las características de la organización y del entorno condicionan, como es natural, la decisión.

Sin embargo, lo que queremos destacar aquí es la necesidad de establecer, como fase previa a la concepción de la base de datos, la organización de la misma y del equipo que va a intervenir en su creación. Ésta es una responsabilidad de la dirección, la cual deberá decidir y aprobar, de un modo formal, la estructura organizativa del equipo, que tendrá a su cargo la creación de la base de datos, así como de la unidad que se encargará posteriormente de su funcionamiento.

Las líneas generales de quién y cómo va a utilizar y actualizar la base de datos, también serán aprobadas por la dirección, y posteriormente el administrador de la base, con el acuerdo de los usuarios, deberá redactar una detallada normativa que regule estos aspectos.

2.4. Plan de trabajo detallado

Obtenida la *conformidad activa* por parte de la dirección para emprender el proyecto, será preciso crear un plan de trabajo detallado en el que se especifiquen las distintas fases, con los plazos y medios que requerirán cada una de ellas.

En general, da mejores resultados prácticos el desarrollo del sistema de forma gradual, sin intentar integrar a la vez todas las aplicaciones en la base; de esta forma, se consiguen varios objetivos:

- a) La propia experiencia va mostrando los errores cometidos y la forma de solucionarlos.
- b) Por otra parte, una evolución, en lugar de una revolución, permitirá la adaptación y formación de los usuarios, tanto informáticos (analistas y programadores) como no informáticos, los cuales no tendrán que enfrentarse bruscamente, y todos a la vez, con un sistema que, al cambiar sus hábitos de trabajo, siempre creará dificultades y despertará recelos.
- c) Se obtendrán resultados prácticos en menores plazos, lo que suele ser muy conveniente de cara a los directivos y a los usuarios.

Hay que tener en cuenta que la anterior afirmación respecto a la conveniencia de una implantación gradual no significa que no haya que hacer un estudio global detallado y a fondo de toda la información que formará parte de la base de datos.

Para realizar esta planificación es muy importante contar con el acuerdo de los usuarios, ya que en varias de las etapas será obligada su participación, por lo que no se puede prescindir de su opinión.

El plan de trabajo detallado ha de ser aprobado por la dirección antes de pasar a la siguiente etapa, y su rechazo puede obligar bien a una reelaboración del mismo o, incluso, a una vuelta a la etapa inicial de estudio de oportunidad, reconsiderando los objetivos, medios y plazos.

En la figura 8.2 se presentan gráficamente estas actividades iniciales de la creación de una base de datos que corresponden a la estrategia del proceso.

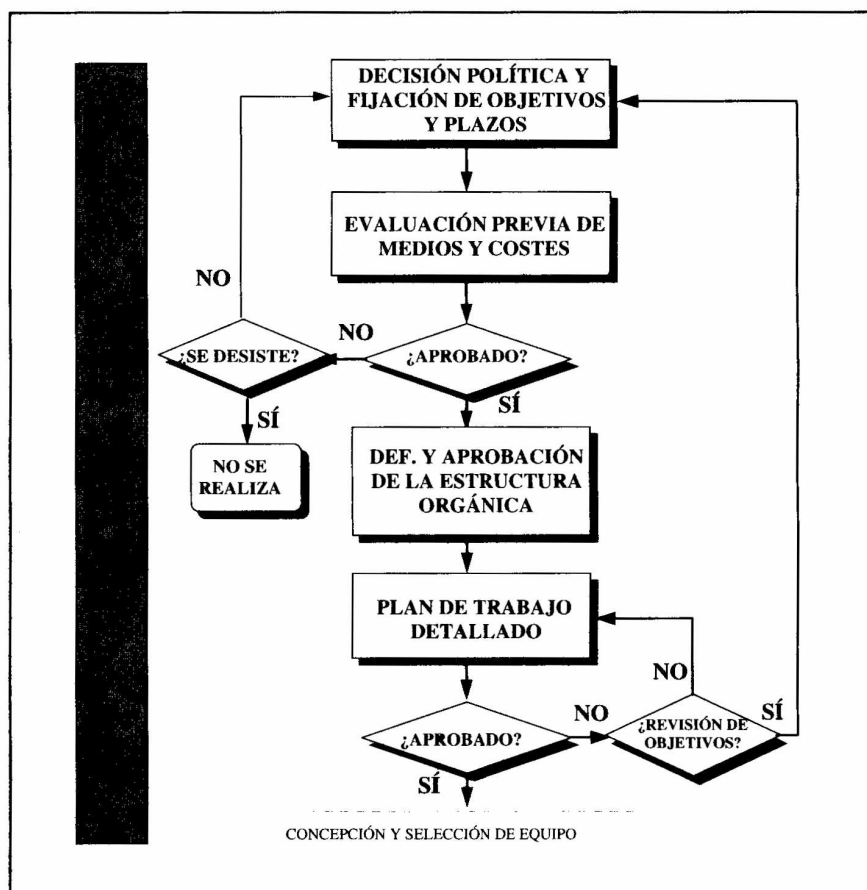


Figura 8.2. Actividades de la fase de estudio previo

3. CONCEPCIÓN DE LA BASE DE DATOS Y SELECCIÓN DEL EQUIPO

En esta fase se realiza un análisis de la información que se ha de integrar en la base de datos a fin de alcanzar los objetivos propuestos y se representa esta información en un modelo conceptual de datos independiente del SGBD que se vaya a utilizar. Además, si no se dispusiese del equipo físico y/o lógico, se ha de llevar a cabo la evaluación y selección del mismo. En la figura 8.3 se representan las actividades que integran esta fase.

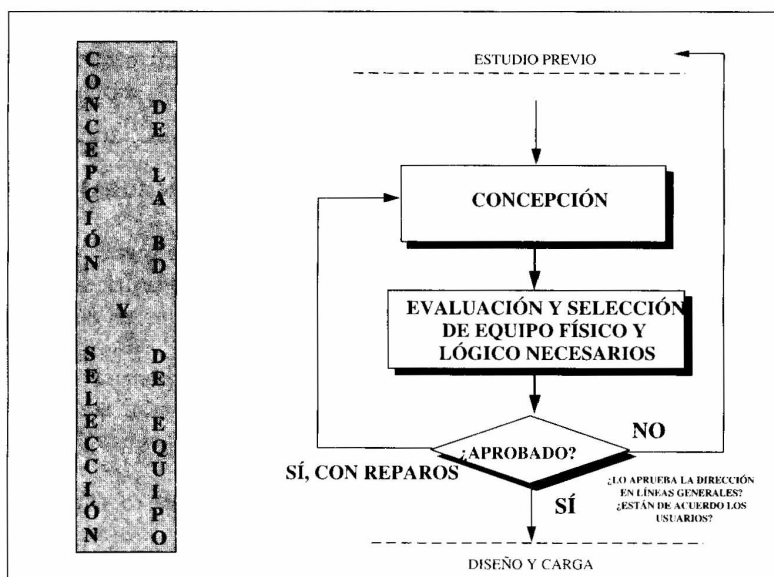


Figura 8.3. Actividades de la fase de concepción de la base de datos y de selección del equipo

3.1. Concepción de la base de datos²

En la fase de concepción se analiza el sistema existente (funciones ya informatizadas, si es que existen) lo que dará una primera imagen, probablemente deformada, del mundo real (empresa u organismo), a continuación se determinarán las necesidades de los usuarios, concretándose las funciones que hay que integrar en la base de datos y las modificaciones que habrá que introducir en las aplicaciones existentes para que se adapten mejor a los fines de la organización y al nuevo enfoque que supone la puesta en marcha de la base de datos.

² En el capítulo 9 se profundiza en el estudio del modelado conceptual.

Mediante estos dos pasos llegaremos a una lista de las informaciones que la organización necesita, así como de los requisitos del sistema, a partir de los cuales se podrán concretar qué datos de entrada, qué procedimientos y qué medios se precisarán para obtener dichas informaciones.

También habrá que describir las actividades de la organización, analizándolas en términos de sistema, de subsistemas y de entorno.

Todo ello nos permitirá determinar, por un lado, las características del sistema (requisitos en cuanto a protección de los datos, flexibilidad, etc.) y de su arquitectura (exigencias de acceso en conversacional, lenguajes, etc.) y, por otro lado, el contenido de la base —datos y metadatos—, con especificación de su volumen, volatilidad, normas de validación, y una lista de reglas de gestión.

En la construcción del esquema conceptual se puede contar con la ayuda del computador que, en un proceso interactivo, irá poniendo de manifiesto las inconsistencias del esquema propuesto por el diseñador, el cual lo podrá ir depurando paso a paso con ayuda de las indicaciones suministradas por la máquina.

La fase de concepción termina contrastando el esquema conceptual que podríamos llamar *bruto*, con la realidad y sometiéndolo a sucesivas adaptaciones, hasta conseguir, en un proceso iterativo, una representación que sea una síntesis de los *esquemas externos* de los distintos usuarios, a partir del cual se debe poder obtener de nuevo los esquemas externos.

Como se puede deducir de todo lo expuesto, esta fase de concepción es totalmente independiente de la máquina donde el sistema será implementado, y su enfoque está dirigido a obtener un análisis de la información ajeno a cualquier consideración que se relacione con las características del computador o del SGBD que se utilizarán, posteriormente, para su puesta en marcha.

3.2. Especificaciones de las necesidades de equipo físico y lógico

Una vez determinados los requisitos y características que el sistema definido en la fase de concepción necesitará para su puesta en marcha, será preciso evaluar las exigencias en cuanto a equipo, en especial respecto al SGBD y al dimensionamiento del computador (memorias principal y secundarias, capacidad de proceso, etc.).

En general, el organismo dispondrá ya de un equipo que será el que se utilice para implementar el sistema (a veces será preciso acudir a una ampliación), por lo que no nos vamos a ocupar de este tema que, por otra parte, no se diferencia, en la metodología a seguir, del problema general de la evaluación, selección y adquisición de equipos.

En cuanto a la selección del SGBD, aunque a veces el problema viene resuelto por condicionantes externos (tipo de máquina, costes, etc.) que obligan a utilizar un **determinado** producto, en general, sí se deberá proceder a realizar un estudio de los SGBD existentes en el mercado, de sus características y de las posibilidades que ofrecen, para poder elegir aquel que mejor se adapte a los requisitos específicos del sistema de información que se está diseñando.

4. DISEÑO Y CARGA

Esta fase comprende tanto el diseño lógico de la base de datos y su codificación, como la carga de los datos y la prueba de los programas. Al igual que las fases anteriores se trata de un proceso iterativo, al final del cual la base de datos puede entrar en explotación. En la figura 8.4 se representan las distintas actividades de esta fase.

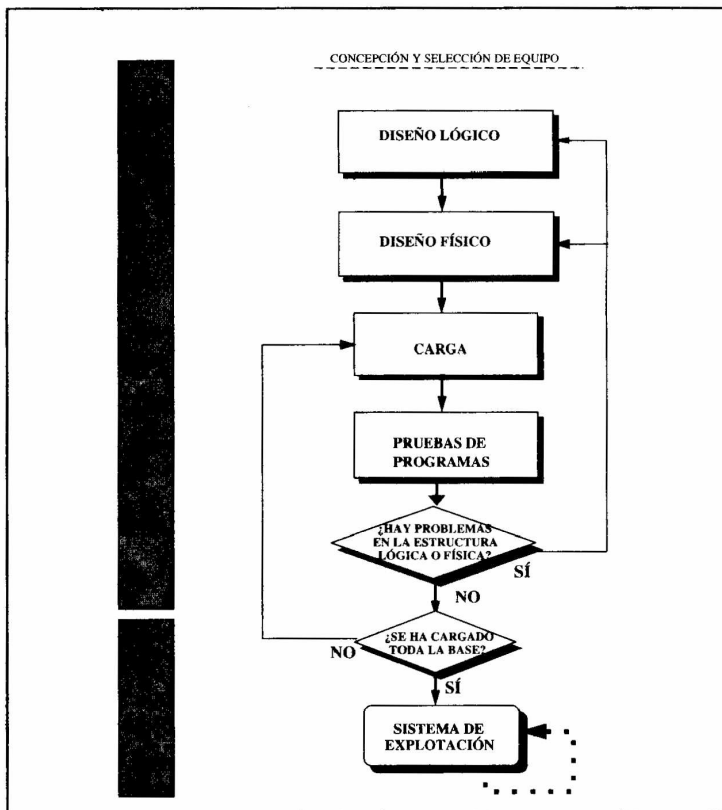


Figura 8.4 Actividades de la fase de diseño y carga

4.1. Diseño lógico y físico³

El esquema conceptual, obtenido en la fase anterior, ha de ser estructurado teniendo en cuenta las peculiaridades del SGBD elegido y de acuerdo con el modelo implementado en el mismo. Definida la estructura lógica de la base de datos se pasará a obtener la estructura física (esquema de almacenamiento o interno).

4.2. Carga y optimización de la base

Definida la estructura física de la base de datos, es preciso cargar los datos en la misma. En general, muchos de estos datos proceden de aplicaciones anteriormente automatizadas, en cuyo caso, lo único que habrá que hacer es proceder a la carga de estos archivos; muchos SGBD dan facilidades para la migración, evitando escribir los correspondientes programas. Para los datos que no se encuentran en soporte de computador, habrá que recogerlos mediante los adecuados impresos e introducirlos en la base de datos.

Al realizar el plan de trabajo hay que contar con esta fase, que puede resultar onerosa, tanto en plazos como en costes, especialmente si los datos no se encontrasen en soporte de computador.

Paralelamente a la fase de diseño se deberán ir desarrollando los programas y procedimientos necesarios para implementar las reglas de gestión que se definieron en la fase de concepción, de forma que, cuando se vayan cargando en la base de datos los distintos conjuntos de información, se puedan ir probando los programas que manejan esos datos.

Cargados en la base algunos archivos, se debe comenzar inmediatamente las pruebas de la base de datos y medir sus rendimientos, con objeto de poder ir ajustando la estructura física e incluso, a veces, la estructura lógica, a fines de optimización.

Existen herramientas capaces de desarrollar prototipos a partir de unas especificaciones iniciales que, si bien no constituyen el sistema de información real, sí pueden ayudar mucho a comprobar si las especificaciones son correctas y mostrar a los usuarios cuál va a ser la interfaz y el comportamiento del sistema.

Si la base de datos ha sido bien diseñada, la independencia entre las distintas estructuras permitirá la optimización de la base por sucesivos retoques sin que ello afecte a los programas de aplicación que acceden a la misma.

³ En los capítulos 10 y 11 se describen de forma detallada las fases de diseño lógico y físico.

Como ya se ha señalado al principio de este capítulo, no suele ser conveniente cargar simultáneamente todos los conjuntos de información que constituirán la base de datos completa; por el contrario, un desarrollo gradual dará, probablemente, resultados más satisfactorios.

5. UNA METODOLOGÍA PARA EL DESARROLLO DE BASES DE DATOS RELACIONALES

Analizadas en líneas generales las distintas fases que lleva consigo la creación de una base de datos, vamos a presentar una metodología para el desarrollo de BD, estudiando en primer lugar el concepto de metodología y sus componentes básicos, para pasar después a considerar la metodología propuesta con sus diferentes fases⁴, y terminar analizando las características que ha de poseer una metodología de desarrollo. Aunque la metodología propuesta podría ser aplicada al desarrollo de una base de datos, sea cual sea el modelo que la soporte, está especialmente orientada al desarrollo en el modelo relacional.

5.1. Concepto de metodología

En los últimos años venimos asistiendo, gracias al avance tecnológico, a una gran difusión de los SGBDR, que ya existen para todo tipo de plataformas, desde los grandes computadores a los computadores personales. Sin embargo, y a pesar del esfuerzo realizado por numerosos investigadores y estudiosos del tema, el desarrollo de una BD sigue siendo una tarea larga y costosa.

Vencer estas dificultades inherentes al desarrollo de una base de datos obliga a abordar dicho desarrollo con procedimientos computador y metódicos, tal como reconocen numerosos autores. Así, en palabras de ROLLAND, FOUCAUT y VENCI (1988), “Las dificultades de la concepción de una base de datos, *claman* por una respuesta metodológica”.

En distintas áreas de la ingeniería del software se han realizado importantes esfuerzos para encontrar las metodologías más adecuadas; esto se debe al gran impacto que una metodología tiene en el desarrollo de un producto software, ya sea en lo que se refiere en los costes y plazos de entrega del mismo, como a la calidad y mantenimiento del producto.

Como señala SOMMERVILLE (1988): “Un buen diseño es la clave de una eficiente ingeniería del software. Un sistema software bien diseñado es fácil de aplicar

⁴ En los capítulos 9, 10 y 11 se profundiza en cada una de las fases de la metodología.

y mantener, además de ser comprensible y fiable. Los sistemas mal diseñados, aunque puedan funcionar, serán costosos de mantener, difíciles de probar y poco fiables”.

A veces el diseño de una BDR se ha limitado simple y llanamente a la teoría de la normalización, cuando en realidad debe abarcar muchas otras etapas que van desde la concepción hasta la implementación.

Por tanto, tenemos que definir en primer lugar qué puede entenderse *por una metodología para el desarrollo de BDR*, cuestión que ha sido abordada por numerosos autores a lo largo de los últimos años. También se han propuesto varias metodologías (como MERISE, REMORA, SSADM, MÉTRICA, DATAID-1, etc.)⁵. A pesar de ello, no existe una metodología de desarrollo de base de datos consagrada, debido quizás a la complejidad del tema, pero sobre todo a la diversidad de opiniones y enfoques existentes en esta área de la ingeniería de software.

WASSERMAN (1979), afirma que “una metodología de diseño puede concebirse como un conjunto de herramientas y técnicas empleadas dentro de un marco organizacional que puede ser aplicado consistentemente a proyectos sucesivos de desarrollo de la estructura de una base de datos”. TEOREY y FRY (1982), definen el diseño de base de datos como “el proceso de desarrollo de una estructura de base de datos a partir de los requisitos de los usuarios”.

Otra definición digna de ser destacada en la de ROCHEFELD (1986), quien señala que “una metodología es una colección de medios propuestos para controlar el proceso de desarrollo”. De forma parecida a la que enuncian SHAN y SHISUAN (1984), al afirmar que “una metodología es una serie de métodos que pueden ser aceptados ampliamente y utilizados en el ciclo de la vida completo del diseño de la base de datos. Estos métodos cumplen distintas tareas en distintos pasos”.

La metodología propuesta en este libro sigue el espíritu de las definiciones anteriores en el sentido de considerar el proceso de desarrollo como un conjunto de medios a aplicar en las distintas etapas del ciclo de la vida de una base de datos. Más precisamente, se ajusta a la definición de Inforsid, posteriormente ampliada en ROLLAND *et al.* (1988), al considerar que: “Una metodología es un conjunto de modelos y herramientas que nos permiten pasar de una etapa a la siguiente en el proceso de diseño de la base de datos”.

Teniendo en cuenta que una metodología es “un conjunto de modelos, lenguajes y otras herramientas que nos facilitan la representación de los datos en cada fase del proceso de diseño de una base de datos, junto con las reglas que permiten el paso de una fase a la siguiente”, el análisis de todos estos elementos es fundamental para poder comprender y aplicar correctamente una metodología de diseño.

⁵ Integrando, casi todas ellas, datos y funciones, con un mayor énfasis en las últimas.

Entendemos por *herramienta* “cualquier recurso particular a disposición de la metodología para realizar las operaciones que en ella se prevén”, BATINI *et al.* (1981); herramientas serán los diagramas, grafos, teorías, etc. que se han de aplicar a las distintas fases del desarrollo. Los modelos, los lenguajes y la documentación son también herramientas, pero dado su especial interés se consideran de forma individualizada.

Ya hemos definido un *modelo de datos* como un conjunto de conceptos, reglas y convenciones que permiten describir y manipular los datos de la parcela del mundo real que constituye nuestro *universo del discurso*; el esquema obtenido al aplicar un modelo de datos a un cierto universo del discurso constituirá la *visión* que del mundo real tiene el diseñador, el cual lo contempla bajo los objetivos impuestos por el sistema de información que está creando.

Un *lenguaje de datos* está siempre basado en un determinado modelo de datos y es el resultado de definir una sintaxis para el mismo, lo que va a permitir expresar un esquema (basado, por ejemplo, en el modelo relacional) en una sintaxis concreta (como, por ejemplo, la del SQL).

La *documentación* nos permitirá describir de forma normalizada los resultados de cada etapa, facilitando así la labor del diseñador y ayudando al mantenimiento de la base.

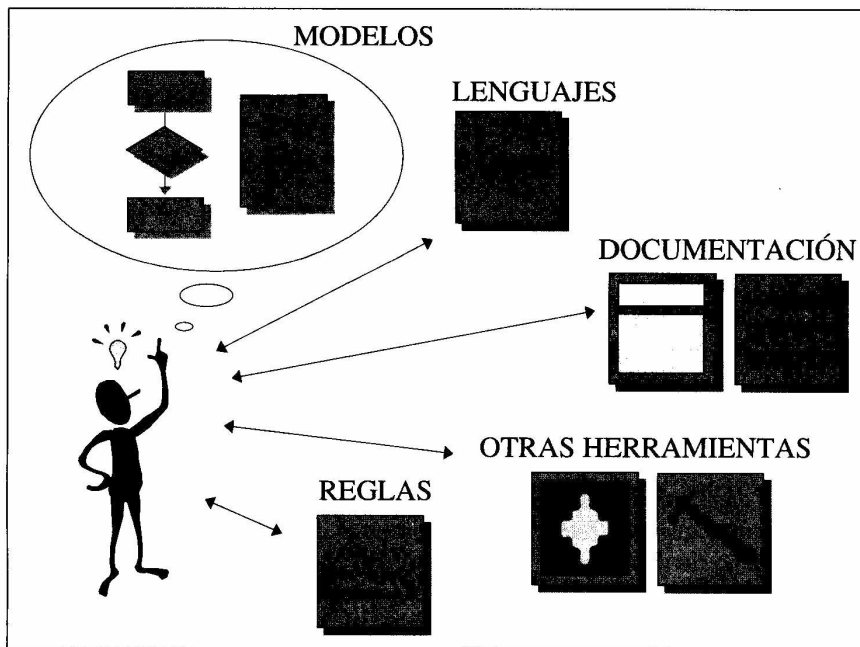


Figura 8.5. Componentes básicos de la metodología

Las reglas actuarán sobre los elementos de entrada en cada fase para conseguir (de manera semiprogramable) las salidas de cada una de ellas, permitiendo en algunos casos elaborar distintas alternativas de diseño.

Estos cinco conceptos (modelos, lenguajes, documentación, otras herramientas y reglas), que se presentan en la figura 8.5, están estrechamente ligados: un lenguaje permite la expresión organizada de los conceptos del modelo, los modelos no pueden aplicarse de forma satisfactoria sin una metodología, y una metodología será más eficaz con el apoyo de herramientas que faciliten su aplicación y con reglas que permitan pasar de una etapa a otra, ayudando a resolver los problemas que van apareciendo en el proceso de diseño, el cual debe estar perfectamente documentado para que puedan llevarse a cabo las revisiones y el mantenimiento. Los participantes (directivos, usuarios e informáticos) constituyen un elemento esencial del desarrollo.

5.2. Enfoque propuesto

La metodología propuesta pretende resolver uno de los principales problemas (si no es el fundamental) del desarrollo de una BD, que no lo constituyen el computador, las teorías o modelos más o menos acertados, sino la comunicación entre las distintas personas que actúan o intervienen a lo largo del proceso. Se trata normalmente de personas con diferentes mentalidades, formación y experiencia que se ven obligadas a trabajar en equipo para desarrollar un sistema útil.

Como se señala en KERSTEN *et al.* (1983), hay dos causas principales que conducen a un diseño incorrecto, que son:

- Falta de conocimiento del dominio de la aplicación; conocimiento que no posee el diseñador informático, pero sí el usuario (aunque no siempre lo tenga bien estructurado ni sepa expresarlo de forma correcta y precisa).
- Falta de experiencia en el modelado: experiencia que sí se le supone al diseñador, pero que el usuario conocedor del dominio de la aplicación no suele poseer.

Para resolver el problema de comunicación entre el usuario y el diseñador, proponemos, al igual que se hace en otras varias metodologías, utilizar un enfoque basado en el ME/R.

Este modelo, sencillo pero, a la vez, suficientemente potente (sobre todo teniendo en cuenta las ampliaciones propuestas que mejoran su semántica), permite entablar un diálogo entre el usuario y el diseñador; diálogo que facilitará que se despejen dudas y aclaren aspectos del universo del discurso a modelar.

En el fondo, el ME/R permite representar conceptualmente los objetos a modelar con un buen nivel de abstracción, o como señalan BRODIE *et al.* (1984), “la popularidad del modelo E/R para el diseño de alto nivel de la BD se debe a la economía de conceptos y a la amplia aceptación de las entidades e interrelaciones como elementos de modelado estructural”.

Permite también este modelo la colaboración de los especialistas con los usuarios; de manera que estos últimos pueden participar activamente como protagonistas del diseño. Como sabemos, esto resulta imprescindible para que la implantación de la base de datos tenga éxito.

Podemos representar esquemáticamente estas primeras etapas de la metodología como se indica en la figura 8.6, en la cual se representa el proceso de diseño para una universidad; el diseñador, partiendo del universo del discurso y apoyándose, en una primera etapa, en el modelo E/R, llega a una estructura relacional (un conjunto de tablas), en la que se almacena toda la información necesaria para la gestión de dicha universidad: alumnos, profesores, departamentos, titulaciones, etc.

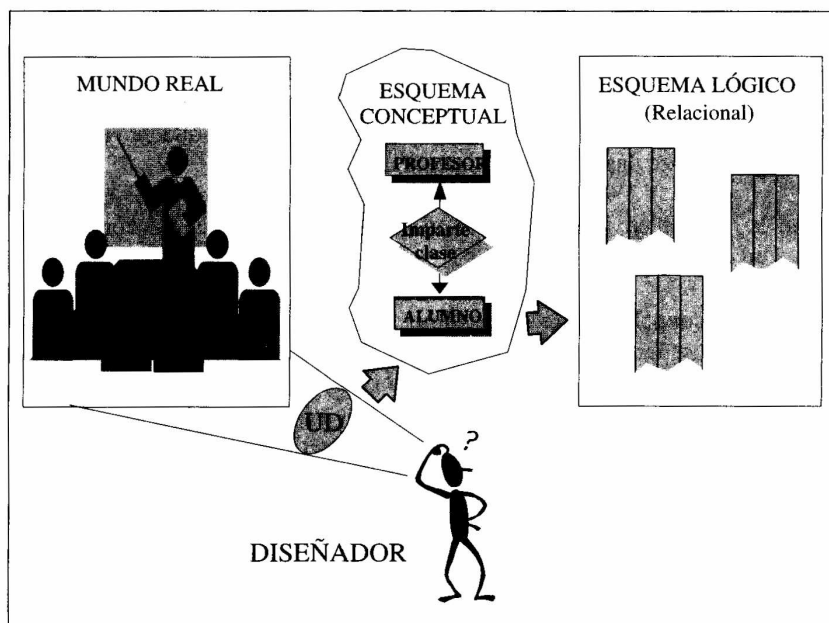


Figura 8.6. Representación esquemática de la metodología propuesta para el desarrollo de una base de datos relacional

En la determinación de las fases de la metodología debemos definir, como señala OLIVÉ (1985), una jerarquía de niveles de abstracción que resulte apropiada, en el sentido de ser lo suficientemente amplia para que a cada nivel le correspondan

decisiones de diseño bien definidas, pero, a la vez, no proponer demasiados niveles, ya que acarrearían muchos conceptos y serían muy sensibles a la interpretación individual de cada diseñador.

Como puede deducirse de un estudio general de varias metodologías existentes, parece que tres grandes fases (que comprenden, a su vez, distintas actividades y tareas), resulta un número apropiado de niveles.

Estas fases, a las que ya nos hemos referido y que se estudiarán con mayor detenimiento en próximos capítulos, son las siguientes:

- **Modelado conceptual:** cuyo objetivo es obtener una buena representación de los recursos de información de la empresa, con independencia de usuarios o aplicaciones en particular, y fuera de consideraciones sobre eficiencia del computador.
- **Diseño lógico:** cuyo objetivo es transformar el esquema conceptual obtenido en la etapa anterior, adaptándolo al modelo de datos en el que se apoya el SGBD que se va a utilizar. Nosotros nos vamos a referir al modelo relacional, pero de forma análoga se podría adaptar esta etapa del diseño lógico a otro modelo de datos, como el Jerárquico o el Codasyl.
- **Diseño físico:** cuyo objetivo es conseguir una implementación, lo más eficiente posible, del esquema lógico.

En el fondo, este enfoque demuestra que la metodología estructura el desarrollo en una “secuencia de pasos de problemas, de modo que cada fase resuelve un problema de diseño bien definido”, HAWRYSZKIEWICZ (1984).

Hay que destacar que cada fase es un proceso iterativo y, como tal, se van produciendo refinamientos sucesivos antes de pasar a las siguientes.

Existe normalmente una realimentación entre las dos últimas fases, ya que pueden producirse cambios en el diseño lógico derivados de requisitos del diseño físico; es decir, muchas veces es preciso *adaptar* el diseño lógico para conseguir una mayor eficiencia del sistema. No es conveniente, sin embargo, que exista realimentación de estos dos últimos niveles hacia el nivel conceptual, ya que éste debe representar los recursos de información de la empresa con independencia de aspectos técnicos.

Las fases del diseño de la BD se pueden relacionar con las clásicas del diseño de un SI (véase figura 8.7).

- El análisis funcional integra el modelado conceptual, en el que, a partir de los requisitos de información, se produce el esquema (o vista) conceptual.
- El diseño (a veces llamado análisis orgánico) integra los diseños lógico y físico de datos.

En una primera fase, a partir del esquema conceptual resultado de la etapa anterior y considerando el modelo de datos en el que se basa el SGBD así como los requisitos de los procesos, se obtiene:

- El esquema lógico global: esto es, el esquema global de la BD, en el modelo Relacional, Codasyl o Jerárquico propio del SGBD.
- Principales vistas de usuario: estructuras externas derivadas del esquema lógico global que resulten de mayor interés en la utilización del sistema.

En una segunda fase, a partir del esquema lógico global y teniendo en cuenta los requisitos de los procesos, las especificaciones del modelo de datos concreto del SGBD que se va a utilizar, así como la configuración y características del equipo físico y del SO, se produce el esquema interno, también denominado por algunos autores *vista del sistema*.

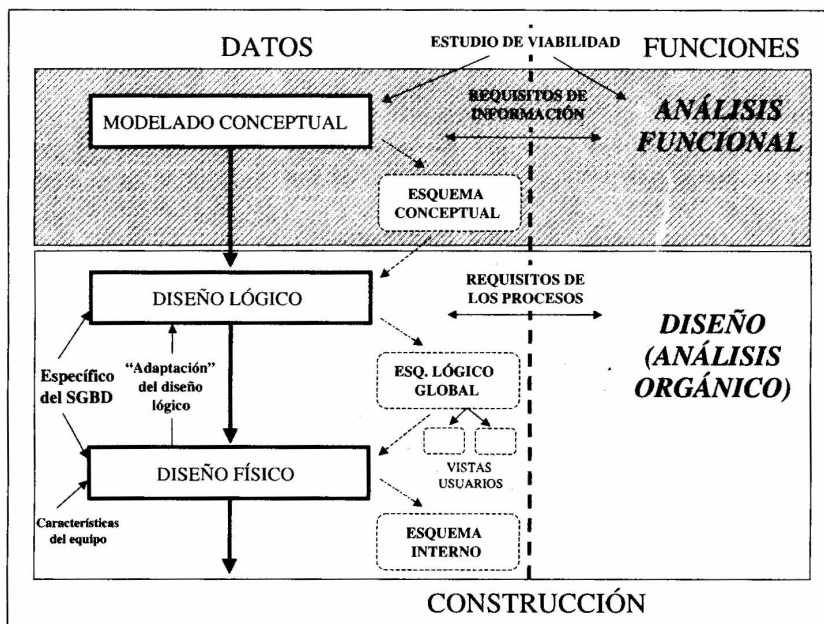


Figura 8.7. Comparación entre el diseño de datos y de funciones

Posteriormente se pasará a la implementación de la base de datos, equivalente a la programación de los procesos (fase de construcción), para pasar a la carga y explotación.

Existen otros enfoques de diseño relacional que no se apoyan en el modelo E/R, sino que llegan directamente al esquema relacional a partir de los atributos considerados aisladamente y de las restricciones semánticas (especialmente dependencias funcionales). La denominada *relación universal*, que contiene el conjunto de atributos y las restricciones semánticas, constituye en este caso el punto de partida de la siguiente etapa de diseño que consiste en la normalización de esta relación.

El enfoque basado en el ME/R, en cambio, da como resultado un conjunto de relaciones (en lugar de la relación universal) que son sometidas al proceso de normalización.

El método, basado en la relación universal, presenta la ventaja de un diseño menos subjetivo, que permite en gran parte aplicar procedimientos algorítmicos. Sin embargo, en él se suele perder más semántica, las relaciones resultantes pueden no corresponder a hechos del mundo real, surgen dificultades para expresar restricciones de integridad referencial y es más difícil que los usuarios participen en el diseño; otro problema que se presenta en este caso es el de recoger la presencia de más de una interrelación entre dos entidades determinadas. Además, los costes de aplicar la teoría de la normalización crecen exponencialmente con el número de atributos por relación; por tanto, si se parte de la relación universal se necesita disponer de herramientas de normalización potentes y sofisticadas que consumen gran cantidad de tiempo y de recursos de máquina.

Nosotros, como ya hemos indicado, concedemos una gran importancia a la participación de los usuarios en el proceso de diseño y pensamos, por tanto, que el ME/R ofrece un mejor punto de partida, ya que se obtienen relaciones más estructuradas, facilita la normalización, y las relaciones finales representan mejor las entidades e interrelaciones del universo del discurso. Un posible inconveniente de este método es que exige cierta práctica en el diseño, pero, en nuestra opinión, sus ventajas superan con mucho este posible inconveniente.

En la figura 8.8 se resumen los dos enfoques expuestos anteriormente. Por un lado (parte izquierda de la figura), partiendo de los atributos y de las restricciones semánticas, se llega a la relación universal, $R \langle A, D \rangle$, donde A es el conjunto de atributos y D el de dependencias; asimismo, se determinan otras restricciones semánticas (como restricciones sobre dominios).

Por otro lado (derecha de la figura), podemos basarnos en el ME/R y llegar a un conjunto de entidades e interrelaciones (con sus correspondientes atributos) y restricciones semánticas; aplicando unas determinadas reglas de derivación (que se

estudiarán más adelante) obtendremos un conjunto de relaciones (denominado $\{R_i\}$ en la figura), cada una de las cuales presenta un conjunto de atributos y dependencias funcionales; además se obtienen otras restricciones semánticas como pueden ser las definidas sobre los dominios o las de integridad referencial. Finalmente, ambos enfoques culminan en el proceso de normalización; aunque, como ya hemos advertido, en el caso de la relación universal la normalización es mucho más costosa, mientras que cuando se parte del ME/R las relaciones están prácticamente normalizadas. Es preciso advertir que ni el conjunto de dependencias funcionales de la relación universal, ni los conjuntos de dependencias de los esquemas de relación cuando se parte del ME/R constituyen, en general, conjuntos mínimos de dependencias, de modo que en ambos casos el proceso de normalización deberá comenzar aplicando algoritmos que calculen un recubrimiento minimal de dependencias.

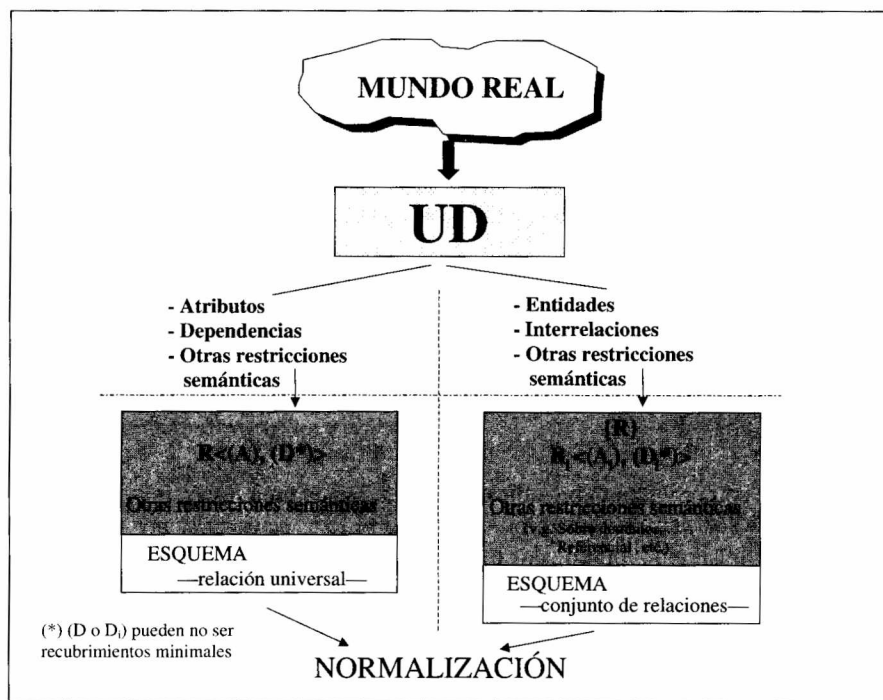


Figura 8.8. Dos enfoques en el desarrollo de una BD

Como señala LAUBER (1982), las distintas herramientas van variando en su grado de formalismo a lo largo de las diferentes fases del ciclo de vida. En la figura 8.9 se puede observar que en las primeras etapas (análisis de requisitos y modelado conceptual) el propósito debe ser la comunicación entre las distintas personas involucradas en el desarrollo, que poseen diferentes tipos de formación y experiencia, mientras que en las últimas fases (diseño lógico, diseño físico e implementación) se

necesita expresar la información de manera procesable por las máquinas, por lo que se ha de utilizar una notación estrictamente formal que no dé lugar a ambigüedades.

	PRIMERAS ETAPAS DE DESARROLLO	ÚLTIMAS ETAPAS DE DESARROLLO
PROPOSITO DE LA NOTACIÓN	Articular ideas y proporcionar comunicación entre personas	Expresar información concebida para su procesamiento por máquina
CARACTERÍSTICAS DE LA INFORMACIÓN	Imprecisa	Precisa, invariable, no ambigua
PERSONAS INVOLUCRADAS	Audiencia variada con diferente formación de base	Programadores, ingenieros de sistemas, entrenados en el uso de lenguajes formales
GRADO DE FORMALISMO	Bajo (preferentemente lenguaje natural)	Estrictamente formal

*Figura 8.9. Distintas características de las etapas de desarrollo de una base de datos.
Inspirada en LAUBER (1982)*

Las distintas metodologías se diferencian en la manera de ir de las primeras a las últimas fases (véase figura 8.10):

- En la práctica suele ser habitual, desafortunadamente, dedicar muy poco tiempo al análisis y modelado conceptual, e incluso al diseño lógico, pasando directamente a implementar tablas en el producto, modificándolas según se vayan identificando nuevas necesidades. Este enfoque, en el que apenas se aplica alguna técnica formal, lleva a unos diseños muy pobres con las consiguientes dificultades de mantenimiento, escasos rendimientos y falta de flexibilidad de los sistemas.
- En el otro extremo podemos encontrar las aproximaciones *puramente* teóricas que preconizan la utilización de lenguajes y técnicas formales casi desde el inicio del proyecto, limitando de esta manera la participación de los usuarios en los proyectos.
- El enfoque de la metodología que aquí proponemos es el de ir adaptando el *rigor* de la notación a medida que progresa el diseño, pensando en cada fase en el tipo de usuarios que se encuentran involucrados.

Con esto se consiguen una serie de ventajas:

- Se requiere menos especialización por parte del diseñador.
- Los usuarios pueden participar en el diseño.
- El diseño es más fácil de verificar por parte de las personas involucradas en el mismo.
- La estructura obtenida es flexible y fácil de mantener.
- El afinamiento físico es más sencillo.
- Cada fase tiene su propia documentación, más o menos formal, según las características de la correspondiente fase.

Las herramientas CASE suelen permitir soportar metodologías más o menos formales, por lo que este enfoque es compatible con la aplicación de este tipo de herramientas.

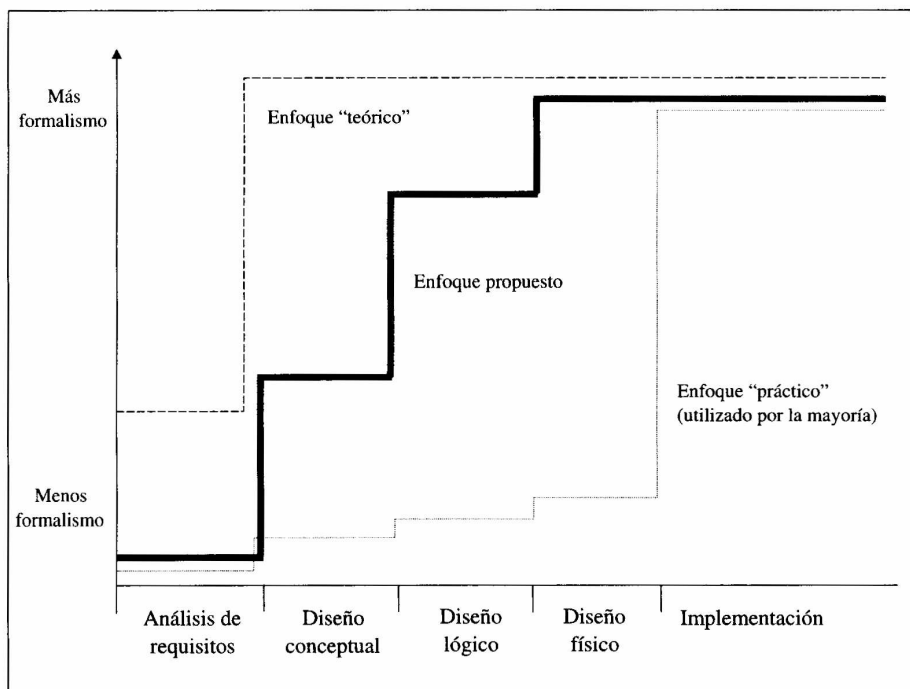


Figura 8.10. Distintos enfoques metodológicos. Inspirada en LAUBER (1982)

En resumen, en la metodología propuesta queremos aprovechar las ventajas de los distintos enfoques. Como señalan FRASER *et al.* (1991), las especificaciones informales tienen la ventaja de la identificación de requisitos, facilidad de aprendizaje y comunicación, mientras que los lenguajes formales proporcionan claridad, precisión y son más adecuados para el análisis y verificación.

5.3. Características de una metodología de diseño

Las características que consideramos deseables en una buena metodología de desarrollo y que creemos que, en mayor o menor medida, reúne la metodología propuesta son las siguientes:

A) Claridad y comprensibilidad

Hemos dicho que, en nuestra opinión, resulta imprescindible que distintas clases de personas (usuarios, técnicos de sistemas, analistas, etc.) participen en el proceso de diseño; por tanto, la metodología debe poseer una sencillez tal que permita que sea explicada a distintos tipos de usuarios.

B) Capacidad de soportar la evolución de los sistemas

Está universalmente admitido que una de las garantías del éxito en el desarrollo de un producto de software es *diseñar y programar para el cambio*. Una buena metodología de diseño deberá ser tal que soporte la evolución del sistema de información sin traumas, produciendo en sus distintas etapas esquemas *evolutivos*, de modo que cuando cambie el universo del discurso sea posible adaptar los esquemas de forma que se recojan dichos cambios sin necesidad de realizar un nuevo diseño completo de la base de datos. Para conseguir este objetivo es fundamental que la metodología proporcione la base para una buena documentación del sistema.

C) Facilitar la portabilidad

El estándar IEEE (1983) considera la portabilidad como “la facilidad con la que un producto de programación puede ser transferido de un sistema informático a otro o de un entorno a otro”. La portabilidad es esencial para conseguir sistemas abiertos. La metodología pretende obtener esquemas portables, para lo cual se utilizan los siguientes recursos:

- Unas etapas de diseño independientes que permiten desviarse en determinados momentos hacia otro tipo de sistemas. Así, aunque la metodología propuesta en esta obra está orientada al modelado relacional, no habría inconveniente en

aplicarla a otro modelo de datos, ya que del esquema conceptual se podría pasar a un esquema en cualquier otro modelo.

- Una subfase denominada *Diseño Lógico eStándar* (DLS), entre el modelado conceptual y el diseño lógico en el SGBDR concreto que se va a utilizar (es decir, en el *Diseño Lógico Específico* o DLE). Esta subfase permite disponer de un esquema relacional específico (como el de ORACLE, DB2, INFORMIX, INGRES, etc.), facilitando así la migración entre diferentes SGBDR o, incluso, entre versiones distintas del mismo producto (como versiones 7 y 8 de ORACLE).

Por otro lado, la portabilidad que ofrecen los propios SGBD comerciales, que suelen trabajar en diferentes plataformas, al actuar muchos de ellos en distintos equipos, facilita también la transferencia de las bases de datos de unos entornos a otros.

D) Versatilidad respecto a tipos de aplicaciones

La metodología propuesta no está orientada a un tipo de aplicaciones concreto, sino que puede utilizarse en aplicaciones diversas, como la gestión de una biblioteca, de un hospital, de una universidad, etc., o para el diseño de bases de datos estadísticas, científicas o de cualquier otro tipo, aunque, en terminados casos, habría que hacer las oportunas adaptaciones.

E) Flexibilidad (Independencia de la dimensión de los proyectos)

Se pretende que la metodología pueda utilizarse tanto en proyectos grandes como pequeños. Para abordar ambos tipos de proyectos se utilizan modelos, herramientas y lenguajes análogos, aunque los proyectos grandes han de complementarse con otras técnicas (como, por ejemplo, la de *integración de vistas*) que se expondrán posteriormente. En cambio, para diseños menos complejos, se pueden simplificar algunas de las etapas de la metodología propuesta, si bien las líneas metodológicas seguirán siendo las mismas.

F) Rigurosidad

Se pretende imprimir un carácter riguroso a los principios metodológicos propuestos. Siempre que ha sido posible (como en el caso de la normalización) nos hemos apoyado en fundamentos teóricos, ya que creemos que la teoría no tiene por qué ir en contra de la práctica. También se ha dado la máxima rigurosidad a las descripciones (ya sean diagramas o lenguajes) a utilizar en el proceso de diseño.

Sin embargo, se ha procurado en todo momento que la metodología no resulte excesivamente *formalista*, ya que un excesivo formalismo puede provocar el rechazo

de determinado tipo de usuarios. Como nos demuestra la experiencia, se puede no ser en exceso *formalista* sin dejar por ello de ser riguroso.

G) Adopción de estándares

Se ha procurado aplicar todos aquellos estándares que para la ingeniería del software en general y para las bases de datos en particular, recomiendan distintas organizaciones internacionales (como ISO, ACM, IEE, etc.). Así, para la descripción del esquema lógico estándar nos hemos basado en el estándar SQL92 de ISO.

Para que una metodología resulte útil y no suponga un *mero bagaje cultural*, debe poderse automatizar (BOTELLA, 1989), aplicando herramientas de tipo CASE (*Computer Aided Software Engineering*) que soporten todas las fases propuestas para el diseño de la BD. En nuestro caso, al utilizar modelos, lenguajes y herramientas muy extendidos (como el ME/R, diagramas de dependencias funcionales, SQL, etc.), la metodología se puede implementar con facilidad en los productos CASE existentes. La documentación debe ser flexible, de forma que pueda almacenarse fácilmente en el computador y extraerse del diccionario o *repositorio* del sistema mediante algún generador de informes.

ENTRADAS Y SALIDAS DEL PROCESO DE DESARROLLO

Podemos considerar que en el proceso de desarrollo de una BD existen una serie de entradas y de salidas que pasamos a resumir:

Entradas:

- **Requisitos de información y objetivos:** que se han especificado al plantearse el diseño de la BD; estos requisitos se obtendrán de las entrevistas con los usuarios, del análisis de los documentos a generar (por ejemplo listados, pantallas, formularios, etc.), junto con los objetivos de la organización.
- **Requisitos de los procesos:** esto es, las distintas características que deben cumplir los programas o aplicaciones que actúen sobre la BD, por ejemplo, en cuanto a tiempo de respuesta.
- **Especificaciones del SGBD:** que incluirán el modelo de datos soportado, además de las características de rendimiento, seguridad, lenguajes, etc. También hay que estudiar los distintos módulos o herramientas que pueden facilitar el diseño lógico y físico de la base de datos. Algunas de estas herramientas (lenguajes de cuarta generación, CASE, etc.) pueden ser proporcionadas también por suministradores distintos de los SGBD.

- **Configuración del equipo físico y del S.O.:** que influirán en mayor o menor medida en el desarrollo de la base de datos, así como en la etapa de diseño físico y ajuste de la misma.

Salidas:

- **Estructuras lógicas de datos:** como resultado del proceso del desarrollo se obtendrá el esquema conceptual, el esquema lógico en el modelo soportado por el SGBD, así como algunas de las principales vistas de usuario que se precisen para interactuar con la BD.
- **Estructura de almacenamiento:** esto es, el esquema interno, donde aparecerán especificados los parámetros y aspectos de diseño físico del sistema, como son, particiones, definiciones de espacio, índices, agrupamientos, etc.
- **Normativa de explotación:** donde se incluirán aspectos de seguridad para la explotación y el mantenimiento de la base.
- **Especificaciones para los programas de aplicación:** para los que se determinan ciertas características a cumplir, especialmente en lo que se refiere al mantenimiento de la seguridad de la base, que no pueden ser recogidas en el esquema.

En la figura 8.11 aparecen las entradas y salidas del desarrollo de una base de datos.

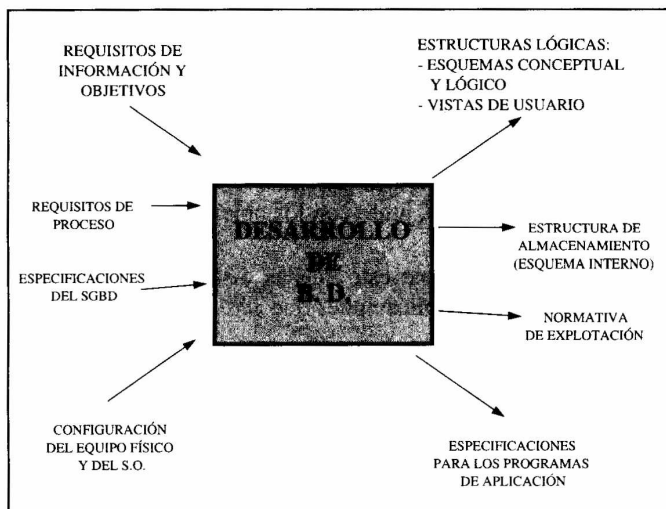


Figura 8.11. Entradas y salidas del proceso de desarrollo de una base de datos

MODELADO CONCEPTUAL

En este capítulo analizaremos la primera fase de la metodología de desarrollo de bases de datos: el modelado conceptual. Empezaremos estudiando el paso del esquema percibido al esquema conceptual, para, a continuación, presentar las distintas etapas de la fase de modelado conceptual, examinando las características que debe poseer un esquema conceptual.

Expondremos también dos tipos de metodologías para el desarrollo de bases de datos, metodologías descendentes y metodologías ascendentes, deteniéndonos en este último caso en el proceso de integración de vistas, que tanta importancia tiene en el área de las bases de datos.

1. ETAPAS DEL MODELADO CONCEPTUAL

El modelado conceptual, también denominado *diseño conceptual*, constituye la primera fase de desarrollo de bases de datos, y puede subdividirse en dos etapas claramente diferenciadas:

A) Análisis de requisitos

Esta primera etapa, en general común para datos y procesos, es la etapa de percepción, identificación y descripción de los fenómenos del mundo real a analizar.

En el análisis de requisitos, como se señala en BENCI y ROLLAND (1979a), se ha de responder a la pregunta: “¿Qué representar?”.

Mediante el estudio de las reglas de una empresa (que proveen el marco para el análisis del sistema) y de entrevistas a los usuarios de los diferentes niveles de la organización (que proveen los detalles sobre los datos) se llega a elaborar un esquema descriptivo de la realidad.

Son varias las propuestas existentes respecto a la forma de expresar el esquema descriptivo, pero en general (y ésta es también nuestra propuesta) se utiliza el lenguaje natural para recoger esta primera información.

Aun cuando esta decisión pueda ser discutible, ya que somos conscientes de los problemas de ambigüedad y escaso formalismo que puede llevar consigo el uso del lenguaje natural; sin embargo, nos parece importante que el usuario pueda establecer en sus propios términos cuál es el problema a resolver.

El planteamiento que presentamos, lo comparten varios autores; así, por ejemplo, FURTADO y NEUHOLD (1986), afirman: "Todos los usuarios entienden un lenguaje común y por eso lo utilizamos como lenguaje de representación de los estados del mundo real...". También MANNINO *et al.* (1983) indican: "lo atractivo del lenguaje natural es que el usuario puede especificar los requisitos sin la intervención de formalismos".

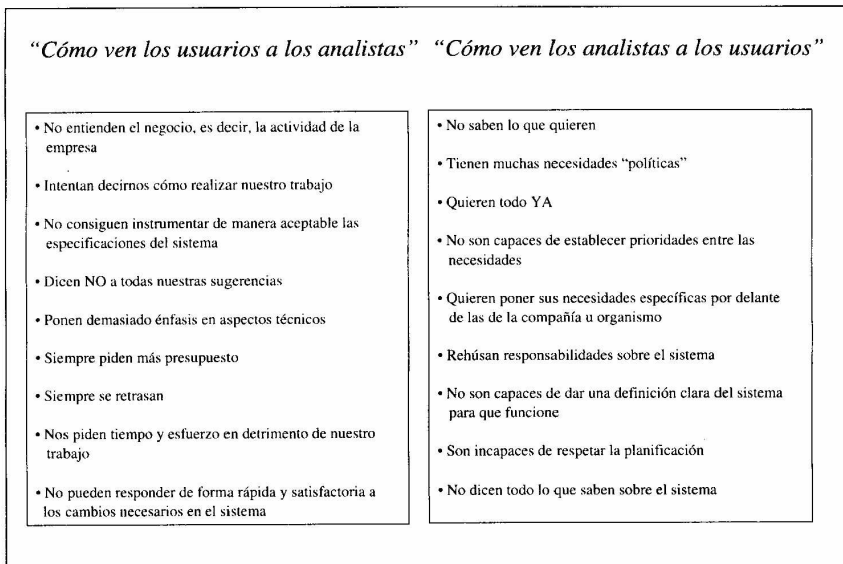


Figura 9.1. Relaciones entre analistas y usuarios, SCHARER (1981)

Ya señalábamos que uno de los problemas más importantes con los que se enfrenta el diseño de una base de datos es el de la comunicación entre las distintas

personas que participan en el mismo, el lenguaje natural servirá para que los usuarios de la base de datos especifiquen fácilmente sus necesidades. En SCHARER (1981) se hace un estudio muy acertado sobre los problemas de comunicación entre usuarios y analistas en la fase de análisis de requisitos. En la Figura 9.1 se reproducen algunas de las actitudes más usuales encontradas en cada grupo con respecto al otro.

Los posibles problemas que presenta esta primera especificación se irán solucionando a lo largo del resto de las etapas de diseño. Podemos afirmar que este primer esquema percibido bruto (como lo llaman BENCI y ROLLAND (1979a)) se irá refinando hasta llegar a un esquema más correcto: el esquema conceptual.

B) Etapa de conceptualización

En ella se transforma este primer esquema descriptivo, refinándolo y estructurándolo adecuadamente. Esta etapa responde a la pregunta: “¿Cómo representar?”. En la figura 9.2 (basada en BENCI y ROLLAND) se recoge el proceso de modelado conceptual, distinguiéndose las dos etapas, así como los distintos procesos que hay que realizar para pasar del mundo real al esquema descriptivo, y de éste al esquema conceptual.

En esta etapa de conceptualización se habrá de buscar una representación normalizada que se apoye en un modelo de datos que cumpla determinadas propiedades, a saber: coherencia, plenitud, no redundancia, simplicidad, fidelidad, etc., para llegar así al denominado esquema conceptual.

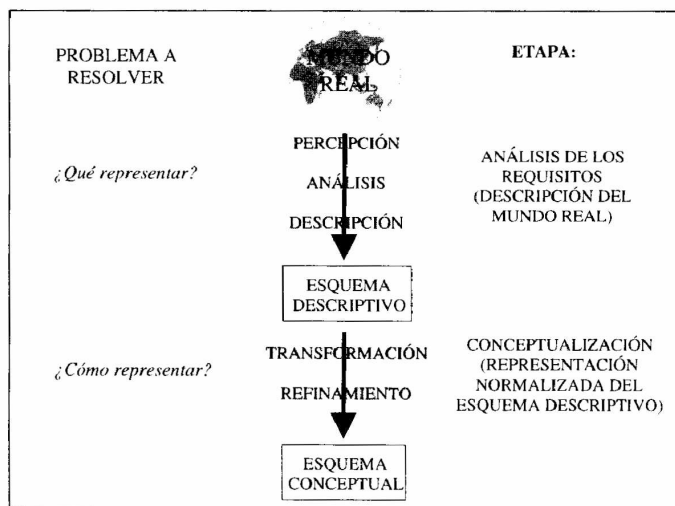


Figura 9.2. Proceso de modelización conceptual

Una característica importante del esquema conceptual, como señala CERI (1983), es que sea *infológico*, en el sentido de que no describa los aspectos ligados a la instrumentación del esquema en un SGBD, sino que permita ver la **información** con todo su contenido semántico.

Como técnica de representación del esquema conceptual, proponemos el ME/R extendido, descrito en el capítulo 2, que puede completarse con una serie de *fichas o plantillas* que sirvan de soporte documental junto al diagrama E/R.

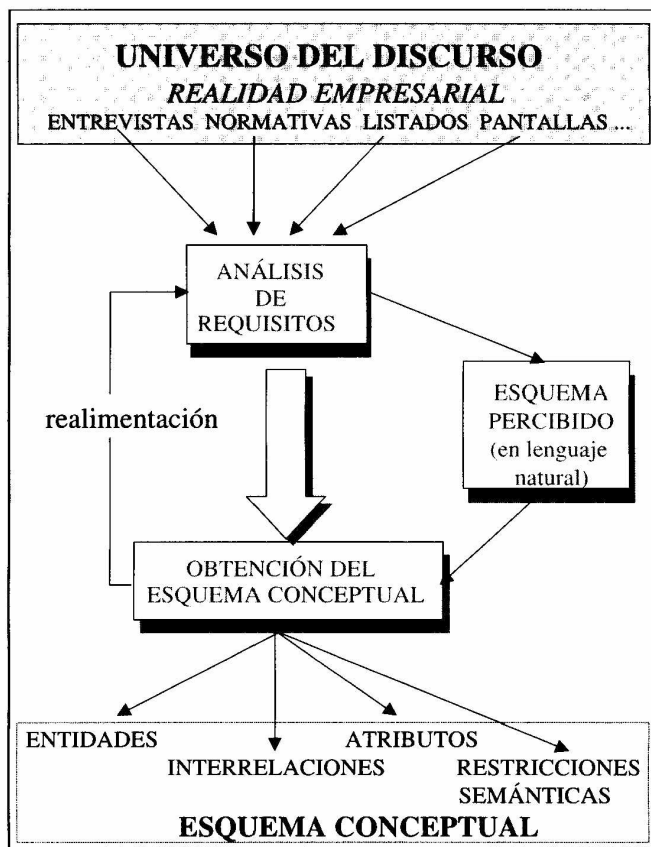


Figura 9.3. Entradas y salidas de la modelización conceptual

En la figura 9.3 pueden observarse, a modo de resumen, las dos fases del diseño conceptual con sus entradas y salidas. Se parte del análisis del universo del discurso (lo que también podría denominarse *realidad empresarial*), analizando los listados, pantallas, normativas, etc. y realizando un conjunto de entrevistas a varios niveles de la empresa.

Posteriormente se elabora un esquema percibido, expresado en lenguaje natural, que nos facilita la obtención del esquema conceptual, esto es, delimita qué entidades, atributos, interrelaciones y restricciones semánticas vamos a considerar.

Este proceso se realiza de forma iterativa hasta que se introducen y clasifican todos los objetos del universo del discurso de forma satisfactoria.

2. PASO DEL ESQUEMA PERCIBIDO AL ESQUEMA CONCEPTUAL

Como hemos señalado, de la primera subfase de la etapa de modelado conceptual se obtiene un esquema percibido en lenguaje natural que representa los requisitos del sistema a diseñar.

Este primer esquema describe lo que deseamos almacenar y resulta del análisis de la documentación existente, junto con las entrevistas a los usuarios. Posteriormente, este esquema se irá refinando sucesivamente y normalizando hasta obtener un esquema en el modelo E/R.

Será preciso, por tanto, *interpretar* las frases del lenguaje natural en el que está descrito el esquema percibido, convirtiéndolas en elementos del modelo E/R, como son las entidades, los atributos y las interrelaciones.

Si bien no existen reglas deterministas que nos digan qué elemento va a ser una entidad o cuál otro una interrelación, sí podemos enunciar unos principios generales que, junto al buen criterio del diseñador, puedan ayudarnos a elaborar un primer esquema conceptual que será sometido después a un proceso de refinamientos sucesivos.

En STOREY y GOLDSTEIN (1993) y RAM (1994) se describen brevemente algunos de los sistemas de diseño de bases de datos basados en conocimiento y se esboza una estructura con los aspectos clave que se deben tener en cuenta en la construcción de sistemas expertos de este tipo. Una de las características esenciales es que estos entornos deben proporcionar un diálogo efectivo con el usuario, sobre todo si están orientados al aprendizaje. Por ello, el lenguaje natural desempeña un papel importante en la comunicación entre sistema y diseñador.

Aunque son muchos los sistemas que intentan aprovechar la información expresada por medio del lenguaje natural, no todos siguen las mismas aproximaciones. Algunos rasgos diferenciadores son:

- Grado en que incorporan conocimiento lingüístico (sintáctico, semántico y pragmático); existen enfoques centrados sólo en palabras clave, otros en análisis sintáctico y otros en análisis semántico.
- Sistemas de representación utilizados, tanto para el conocimiento extraído del lenguaje natural como para el conocimiento subyacente al modelo de datos.
- Cobertura (cuáles son los fenómenos lingüísticos, tanto sintácticos como semánticos, que permiten en las oraciones que procesan; si manejan pseudo lenguaje natural o no).
- Robustez (si el sistema es capaz de realizar interpretaciones parciales en caso de no poder analizar una oración o texto completos o si es capaz de proponer varias alternativas en caso de tener información incompleta).
- Diálogo con el usuario, relacionado con el aspecto anterior, pues la interpretación puede ser automática o semi-automática dependiendo del grado de interacción con el usuario.
- Grado de interacción con otras técnicas (como herramientas gráficas, tutores interactivos) que ayudan a la construcción de los esquemas.

2.1. Enfoque para el análisis de requisitos

En MARTÍNEZ (1998), se recogen los diferentes enfoques empleados para el análisis de requisitos textuales mediante técnicas de análisis del lenguaje. Expondremos aquí el enfoque lingüístico de CHEN (1983), pionero en el modelado conceptual; el enfoque de *categorización de objetos* de CARSWELL y NAVATHE (1983); y las propuestas más actuales de herramientas que intentan (semi) automatizar el proceso de construcción de esquemas conceptuales de base de datos a partir de especificaciones textuales.

En el enfoque lingüístico CHEN (1983), se proponen un conjunto de heurísticas que tienen en cuenta tanto la estructura de las oraciones como los atributos gramaticales de las palabras. El objetivo de estas recomendaciones es depender menos de la intuición de los diseñadores y más de métodos estructurados. El autor presenta 11 heurísticas (no reglas, ya que para cada una de ellas se pueden encontrar contra ejemplos). Algunas de ellas son:

- Un sustantivo (nombre común) que actúa como sujeto o complemento directo en una frase es, en general, un tipo de entidad, aunque podría ser un atributo. Por ejemplo, en la frase “Los estudiantes solicitan becas”, existen dos posibles entidades: ESTUDIANTE (sustantivo que actúa como sujeto) y BECA (que actúa como complemento directo).

- Los nombres propios nos suelen indicar ejemplares de un tipo de entidad; por ejemplo, “Hens, R.” indica un ejemplar de ESTUDIANTE.
- Un verbo transitivo o una frase verbal es un tipo de interrelación, en la frase anterior “solicitar” indica una interrelación entre las dos entidades, ESTUDIANTE y BECA.
- Una preposición o frase preposicional entre dos nombres suele ser un tipo de interrelación, o también puede establecer la asociación entre una entidad y sus atributos. Por ejemplo, al decir, “el área del departamento”, bien podemos estar indicando la interrelación entre las entidades DEPARTAMENTO y ÁREA, o bien podemos estar asociando el atributo *área* a la entidad DEPARTAMENTO.

Por tanto, basándonos en conceptos lingüísticos podemos llegar a perfilar un **primer** esquema conceptual.

Otro acercamiento válido al problema de la categorización de los objetos es el que presentan CARSWELL y NAVATHE (1983), quienes afirman:

- Una entidad es un *objeto de datos* que tiene más propiedades que su nombre o se utiliza como operando en una sentencia de selección, borrado o inserción. Por ejemplo, en la universidad existen profesores que poseen una serie de propiedades, como son el nombre, apellidos, DNI, dirección, etc. PROFESOR es una entidad, ya que tiene unas propiedades (nombre, apellidos, etc.). Otro ejemplo, cuando un ESTUDIANTE deja de serlo es preciso darle de baja de la base de datos; ESTUDIANTE es una entidad, por ser un operando en una sentencia de borrado.
- Un atributo es un *objeto de datos* al que se le asigna un valor o se utiliza como operando en una operación aritmética, booleana o tira de caracteres. Por ejemplo, se puede consultar si el nombre de un profesor es *Paloma*, por lo que *nombre* es, según este enfoque, un atributo.
- Una interrelación es un “objeto de datos” que hace posible la selección de una entidad por medio de una referencia a un atributo de otra entidad; así, por ejemplo, podemos seleccionar los profesores que pertenecen a una determinada área, por lo que “**pertenecer**” es una interrelación, ya que nos permite seleccionar una entidad (PROFESOR) por medio de una referencia a un atributo de otra entidad (*Nombre de área*).

En definitiva, se trata de reglas basadas en el papel o rol que un determinado **objeto** desempeña en el proceso de información.

Por tanto, combinando ambos enfoques, si decimos que "... la universidad tiene un conjunto de profesores de los que interesa recoger, además del nombre y apellidos, el DNI, la dirección, materia que imparten y tipo de profesor..."; basándonos en el primer enfoque, podemos decir que PROFESOR, al ser un nombre o sustantivo que actúa como complemento objeto, es un tipo de entidad (o de atributo); según el segundo enfoque, se le puede catalogar definitivamente como entidad, ya que PROFESOR es un *objeto de datos* que tiene más propiedades que su nombre y apellidos (DNI, dirección, etc.) y que se utilizará en sentencias de borrado, inserción y modificación al dar de baja o de alta distintos profesores.

Si se dice que "Mercedes García ha solicitado la beca para estudios de postgrado, estos nombres propios nos están indicando ejemplares de entidades (de la entidad ESTUDIANTE el primer nombre propio, y de la entidad BECA el segundo).

La titulación del estudiante es un *objeto de datos* al que se le asigna una valor (como Ing. Informática, Ing. Telecomunicaciones, etc.) y que actúa como operando en una operación booleana o de tira de caracteres. Podrá ser usual, por ejemplo, consultar los estudiantes de doctorado que son Ingenieros en Informática, o si un determinado estudiante es Ingeniero de Telecomunicaciones, etc. En el enfoque lingüístico, la proposición nos asociaría el atributo titulación a su entidad (ESTUDIANTE).

Si decimos que "... Los departamentos preparan programas de doctorado...", podemos ver como **preparar** es una interrelación entre departamentos y programas, ya que es un verbo transitivo (primer enfoque), también se puede considerar (segundo enfoque) como un *objeto de datos* que hace posible la selección de una entidad (PROGRAMA) referenciando un atributo de otra entidad (DEPARTAMENTO), por ejemplo, al consultar los programas preparados por el departamento de Lenguajes y Sistemas, Informática e Ingeniería del Software.

El estudio de las frases que definen los requisitos del sistema permite ir clasificando los distintos objetos. Interés especial presentan dos verbos muy comunes en la especificación de los requisitos: **ser** y **tener**, como estudian STOREY y GOLDSTEIN (1988).

- 1) "**ES UN**" nos permite, como ya hemos visto, crear jerarquías de entidades, de hecho corresponde al concepto de generalización de SMITH y SMITH (1977).

Un ejemplo de este tipo de interrelaciones puede ser el siguiente: "...Tanto un doctor como un no doctor de nuestra base de datos son profesores...", en consecuencia podemos establecer una jerarquía tal como aparece en la Figura 9.4. Los atributos de PROFESOR los heredarán tanto DOCTOR como NO DOCTOR. Por tanto, en la entidad PROFESOR se encontrarán los atributos comunes a las entidades DOCTOR y NO DOCTOR; sin embargo,

puede haber atributos que sean específicos de cada una de ellas y que no aparezcan, por tanto, en el supertipo; por ejemplo, *fecha doctorado* podría ser un atributo de DOCTOR, pero no la tenemos para NO DOCTOR.

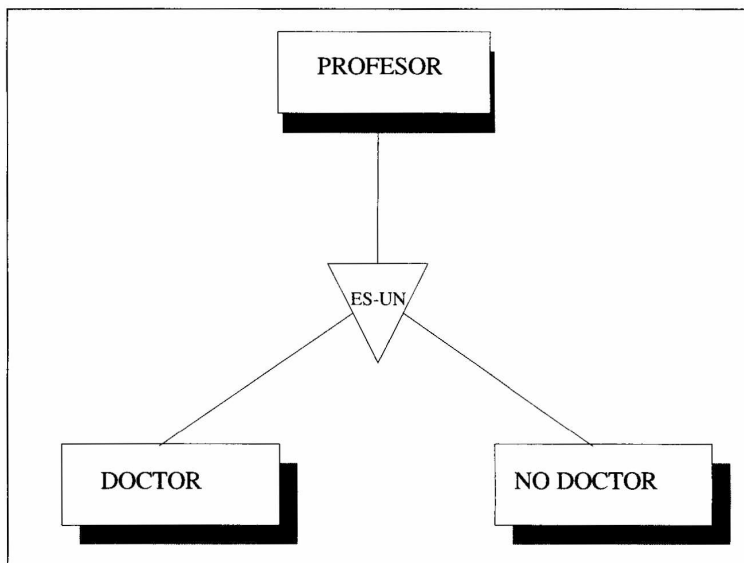


Figura 9.4. Ejemplo de interrelación "es-un"

- 2) **"TIENE"**: Este verbo, sobreutilizado en castellano, posee múltiples interpretaciones, que pueden ser más o menos específicas según la acepción del verbo en la correspondiente frase:
- **Interrelación general entre entidades:** en cuyo caso el verbo se utiliza como otro cualquiera, sin una acepción específica; por ejemplo, "... los alumnos *tienen* un tutor..." nos establece la interrelación entre las entidades ALUMNO y TUTOR, donde, *tener* actúa de forma totalmente análoga a cualquier verbo transitivo, y podría ser sustituido, por ejemplo, por *asignar*.
 - **Asociación de las entidades con sus atributos;** por ejemplo, si decimos que "... los profesores *tienen* un nombre y apellidos, un DNI, una dirección y un teléfono...", estamos asociando a la entidad PROFESOR una serie de atributos: nombre, apellidos, DNI, dirección, teléfono.
 - **Agregación de entidades para formar una entidad compuesta** (corresponde a la abstracción de agregación); por ejemplo, el CURSO *tiene* una PARTE_TEÓRICA y una PARTE_PRÁCTICA donde CURSO

es el elemento *compuesto* y PARTE_TEÓRICA y PARTE_PRÁCTICA son los elementos *componentes*. Podríamos haber sustituido el verbo *tiene* por *estar compuesto* (o *es un agregado*).

- “**Dependencia en identificación (o en existencia)**¹”: así, podemos decir que “... un curso de doctorado *tiene* varias ediciones...”, en el sentido de que una edición es un ejemplar de un curso de doctorado. En este caso el identificador de la entidad que es ejemplar (EDICIÓN) se suele formar, como ya hemos indicado, con el identificador de la entidad principal (en el ejemplo, CURSO) junto a un atributo discriminante de la ocurrencia. Podríamos pensar en identificar las ediciones de cursos con cinco dígitos 99999 y a los ejemplares correspondientes concatenarles un discriminante de dos dígitos, con lo que quedarían identificados como 99999-01, 99999-02, etc.

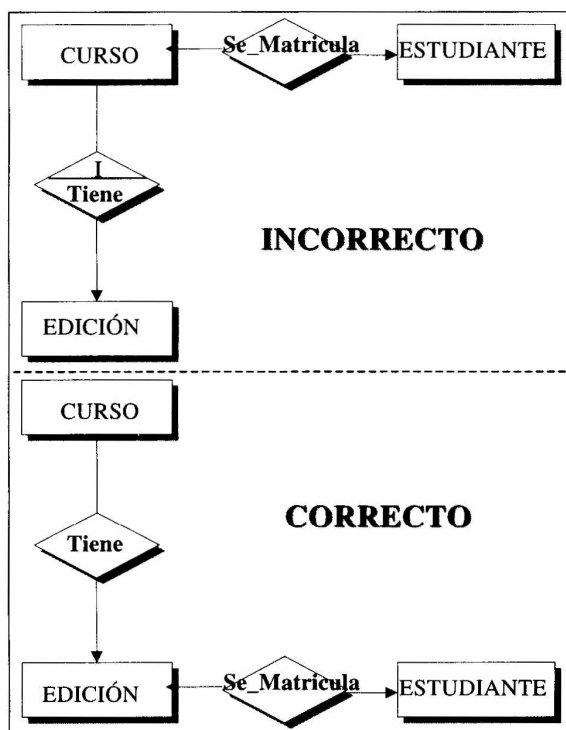


Figura 9.5. La formación de jerarquías de entidades puede obligar a la redefinición de interrelaciones

¹ El razonamiento lo vamos a realizar respecto a la dependencia en identificación, pero sería análogo para el caso de la dependencia en existencia.

La aparición de jerarquías “ES_UN” como de jerarquías de agregación o de identificación pueden obligar a revisar con atención, después de definida la jerarquía, si otras interrelaciones entre las entidades que forman la jerarquía se han especificado al nivel adecuado. Por ejemplo, en la figura 9.5, si se tenía la interrelación **Se_Matricula** entre CURSO y ESTUDIANTE y, posteriormente, aparece la entidad EDICIÓN con una dependencia en identificación respecto a CURSO, la interrelación **Se_Matricula** habría que redefinirla diciendo que “... los estudiantes se matriculan en ediciones de cursos...”; es decir, la interrelación no está entre las entidades ESTUDIANTE y CURSO, sino entre ESTUDIANTE y EDICIÓN.

Análogamente, se tendrán que revisar los atributos de las entidades que pertenecen a una jerarquía, ya que deben asociarse a la entidad de nivel más adecuado semánticamente, esto es, a la de nivel más alto posible. Así, por ejemplo, en la figura 9.6 puede observarse que el nombre y el DNI son comunes a las entidades DOCTOR y NO DOCTOR, por lo que es más correcto definirlas en la superentidad PROFESOR.

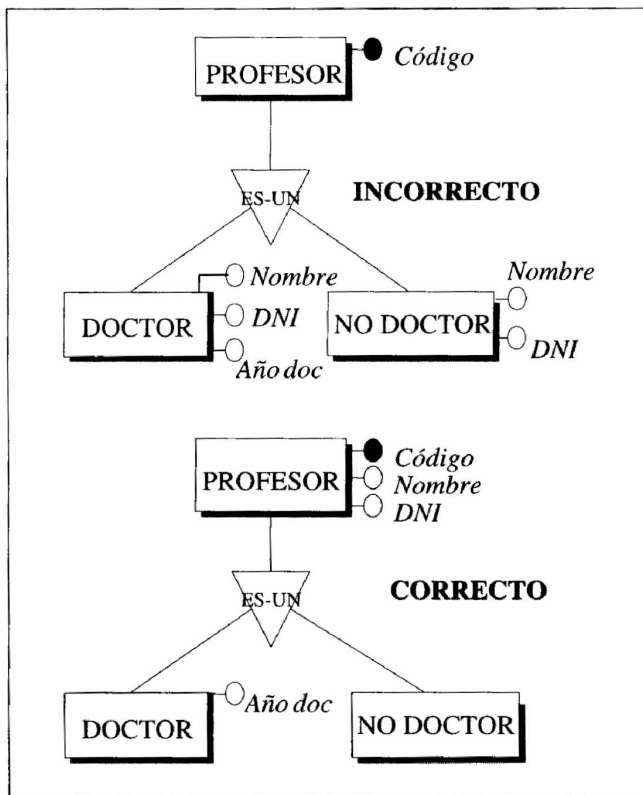


Figura 9.6. La formación de jerarquías de entidades puede obligar a la redefinición de atributos

Existen otros aspectos a tener en cuenta, entre los que cabe destacar que del número de las entidades (singular/plural) puede deducirse ciertos tipos, cardinalidades y grados de las interrelaciones; así, si decimos "... un estudiante se matricula en una o varias ediciones de cursos... y ... en una edición de un curso se matriculan varios estudiantes..." podemos deducir que la interrelación es de tipo N:M, y de grado 2.

Dijimos al comienzo de este apartado lo difícil que puede resultar la categorización de los objetos del universo del discurso. Hemos visto, sin embargo, unos principios generales que pueden servirnos como guía en este proceso, aunque siguen subsistiendo problemas y ambigüedades, como la distinción entre atributo y entidad, que no siempre resulta clara del análisis lexicográfico del esquema percibido. Hagamos, siguiendo a BRATHWAITE (1985), algunas consideraciones que nos pueden ayudar a decidir si es mejor incluir un *objeto de datos* como atributo o como entidad interrelacionada con la entidad de la que se supone que podría ser atributo.

Es preferible considerar el *objeto de datos* como entidad, en lugar de como atributo, en los siguientes casos:

- Si el *objeto de datos* tiene por sí mismo asociados otros atributos, por ejemplo, si la materia que imparte un profesor (que considerábamos un atributo de PROFESOR) tiene a su vez otros atributos, como número de temas, horas de práctica, horas de teoría, etc.), conviene crear la entidad MATERIA.
- Si el *objeto de datos* estuviese relacionado con otras entidades: por ejemplo, si el área la hubiéramos considerado como un atributo de PROFESOR, no podríamos reflejar las posibles interrelaciones existentes entre las áreas y los departamentos; por ejemplo, para especificar que el departamento de Informática se compone de las áreas de Lenguajes y Sistemas Informáticos y de Ciencias de la Computación e Inteligencia Artificial.

Estas reglas que acabamos de exponer pueden servirnos de ayuda en el paso del esquema percibido al conceptual. Sin embargo, no se debe olvidar que se trata de un proceso iterativo, y que sólo mediante refinamientos sucesivos, a lo que nos ayudará la crítica constructiva de los usuarios, podremos llegar a un esquema conceptual que refleje lo más fielmente posible la estructura de la información de la empresa u organismo para el que estamos realizando el diseño de la base de datos.

2.2. Creación de esquemas conceptuales a partir de especificaciones textuales

En cuanto a los entornos que automatizan la tarea de adquirir esquemas conceptuales de BD a partir de especificaciones textuales, los primeros sistemas desarrollados, SECSI (BOUZEGHOUB y GARDARIN, 1984), Modeller (TAUZOVICH, 1990), NIAM (Natural Language Information Analysis Method)

(DUNN y ORLOWSKA, 1990), OICSI (ROLLAND y PROIX, (1992), etc. siguen una aproximación sintáctica basada en la identificación de ciertos patrones lingüísticos. Estos sistemas emplean reglas sintácticas (gramáticas) del lenguaje natural de una manera bastante *ad hoc* (del tipo <Grupo Sujeto> <Grupo Verbal> <Grupo Complemento>) para después asignar los distintos constructores de un modelo conceptual (entidades, atributos, interrelaciones, etc.) a los distintos componentes de una oración según el tipo de verbo de la oración, la categoría gramatical de las palabras, etc. Su principal problema consiste en que no contienen en realidad bases de conocimiento lingüísticas sobre las palabras (como léxicos, por ejemplo) necesarias para poder llevar a cabo un análisis profundo del lenguaje.

Otras propuestas más actuales tratan de salvar este obstáculo de la falta de conocimiento lingüístico. El sistema NL-OOPS, (MICH, 1996), soporta la fase de análisis de requisitos, generando modelos orientados al objeto, incluso según distintas metodologías, utilizando LOLITA (Large-scale, Object-based, Linguistic Interactor, Translator, and Analyser), MORGAN *et al.* (1995). LOLITA es un sistema de Procesamiento de Lenguaje Natural (PLN) a gran escala basado en una red semántica que puede analizar texto morfológica, sintáctica, semántica y pragmáticamente.

Brevemente, la especificación de requisitos se procesa para conseguir un conjunto coherente de requisitos (corrección de textos, eliminación de inconsistencias, etc.), los cuales se analizan completamente en LOLITA. Los resultados del análisis se añaden a la base de conocimiento del sistema PLN, es decir, la red semántica de LOLITA denominada SemNet. SemNet es una representación particular de un grafo conceptual con más de 100.000 nodos conectados que representan tanto la información léxica como semántica.

COLOR-X (Conceptual Linguistically-based Object-oriented Representation language for Information and Communication Systems), BURG (1997), propone un método para análisis de requisitos dentro del proceso de desarrollo software. La base lingüística de COLOR-X se refleja en dos aspectos. El primero concierne a la definición de un lenguaje de modelado formal denominado Conceptual Prototyping Language (CPL) muy próximo al lenguaje natural y basado en la lógica y en la teoría lingüística de la Gramática Funcional. El segundo está relacionado con el proceso de análisis de requisitos, que transforma un documento textual en un esquema conceptual siguiendo un proceso guiado por la semántica de las palabras y que emplea una base de datos léxica que permite recuperar los significados de las palabras así como verificar combinaciones correctas de palabras.

Por ejemplo, la especificación “When a company has sold a car to a customer, it has to send the bill to this customer within a week” tiene la siguiente sintaxis en CPL:

MUST: ACTION:

```
send(ag= C in company) (pat= bill) (rec =C2 in customer) (temp=T2 in time)
  (id: T2 =T1 + 1* week)
  (sit: PERF: sell(ag=C in company) (pat=car) (rec=C2 in customer)
    (temp= T1 in time))
```

donde los *roles* (ag –agente-, pat –paciente-, rec –recipiente- y temp –tiempo-) se corresponden con las funciones semánticas y algunos complementos modales de la Gramática Funcional. La *modalidad* (por ejemplo, MUST significa *debería*) y el *tiempo* (por ejemplo, ACTION indica presente, etc.) son también nociones lingüísticas que aclaran el estado de la interrelación especificada.

El lenguaje de especificación CPL es la base del modelo conceptual CSOM (COLOR-X Static Object Model) así como del modelo de eventos CEM (COLOR-X Event Model).

El método para el análisis semántico del LN consiste en cuatro pasos:

- Paso 1:** Seleccionar las palabras y oraciones del documento de requisitos que son relevantes para los modelos de COLOR-X marcándolas con información sintáctica sencilla (nombre, verbo, etc.). Por ejemplo, la identificación del *dominio de ventas* puede conducir a la identificación de *vendedor, clientes, productos, precios*, etc.).
- Paso 2:** Descomponer las oraciones complejas y/o combinar las que sean redundantes o se solapen en otras más comprensibles que puedan especificarse en el lenguaje CPL.
- Paso 3:** Marcar las palabras seleccionadas con información semántica y sintáctica adicional recuperada del léxico. El léxico proporciona una lista de posibles significados, por ejemplo, los significados de la palabra *lend*, recuperados de la base de datos léxica *WordNet*, MILLER (1995), son:

1. give temporarily
2. add a quality
3. have certain characteristics of qualities for something

Wordnet proporciona además los esquemas en los que ocurre un verbo y éstos se podrán traducir a estructuras CPL. Por ejemplo, para el significado 1 se tienen los esquemas:

- a. somebody ---s somebody something
- b. somebody ----s something to somebody

La estructura CPL del esquema 1 sería:

lend(**ag** = library) (**go** = book) (**rec** = member)

Las ventajas de esta formalización son facilitar la identificación de objetos así como la definición del número de objetos involucrados en un evento y el papel que desempeña.

Paso 4: Transformar las oraciones estructuradas en especificaciones CPL y/o representaciones gráficas, como muestra la figura 9.7.

En BURG (1997), se describe el entorno CASE ideal para automatizar el método antes descrito.

Por último, el trabajo de MARTÍNEZ (1998) presenta una aproximación a la extracción (semi) automática al modelado conceptual de BD a partir de textos, que también hace uso de fuentes de conocimiento lingüístico empleando distintos tipos de tecnología de procesamiento del lenguaje natural, pero a diferencia de NL-OOPS y COLOR-X, sin fijar una secuencialidad a priori para el análisis del lenguaje.

El objetivo es permitir una interpretación parcial dirigida por una serie de “pistas” incluidas en el propio texto denominadas *perspectivas lingüísticas*, por ejemplo, patrones o expresiones típicas de los textos del dominio, determinados verbos con preferencias semánticas, etc. Las perspectivas lingüísticas son conjunto de estrategias de interpretación del lenguaje que combinan las distintas fuentes de conocimiento lingüístico (morfología, sintaxis y semántica) según la información disponible más prometedora para guiar la adquisición de conocimiento.

Estas perspectivas guían el proceso de interpretación una vez se ha realizado un análisis superficial de la oración (análisis morfológico y sintáctico parcial). Las perspectivas basadas en sintaxis utilizan, como punto de partida, el conocimiento relativo a la colocación de los elementos en una frase, mientras que las basadas en la semántica parten del significado preferente de algunos términos. Veamos, a continuación, algunos ejemplos.

Existe una perspectiva denominada *patrones de estilo* que describen diferentes estructuras sintácticas típicas de los textos del dominio que responden a convenciones utilizadas para representar significados particulares (generalmente atributos o ocurrencias de una entidad). Generalmente contemplan varios fenómenos de elipsis de verbos. Esta perspectiva conduce la interpretación examinando en primer lugar la sintaxis (patrón) para, a continuación, examinar los atributos morfológicos y

semánticos de los elementos que componen el patrón. En las siguientes oraciones se muestran en cursiva las partes de las oraciones que equiparan este tipo de patrones:

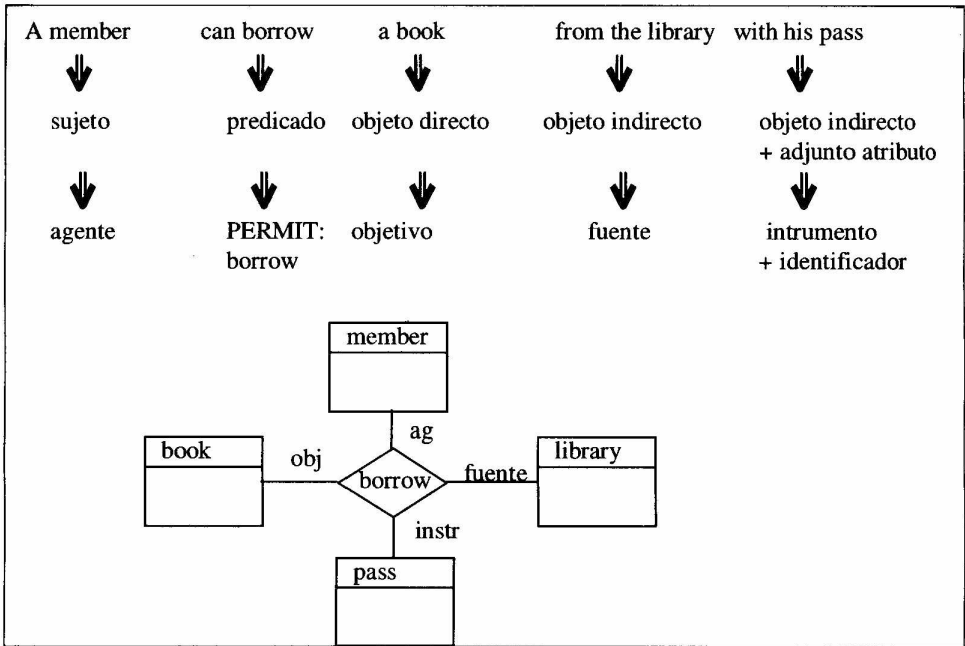


Figura 9.7. Ejemplo de Asociación de Palabras y conceptos CPL

“En una película pueden participar *varios actores (nombre, nacionalidad, sexo)*”

“Un *delincuente (DNI, nombre, teléfono)* es arrestado por uno o varios policías”

“Una película se caracteriza por su título, nacionalidad, productora y fecha (por ejemplo, “*Quo Vadis*”, “*Estados Unidos*”, “*M.G.M.*”, 1955)”

La perspectiva de las *palabras clave* la componen un conjunto de palabras, o secuencias de palabras entendidas como una unidad, que son propias del dominio de BD (terminología específica) con una clara correspondencia con algunos conceptos del mismo. En este caso se hace uso de las preferencias léxicas de las palabras según el dominio en que se utilizan. Por ello, el análisis mediante esta perspectiva está dirigido por la semántica. Las siguientes oraciones muestran algunos ejemplos:

“Cada sala se identifica por un *código de sala*”

“Las asignaturas tienen un *código identificador*, un *nombre* y un *curso*”

“Los documentos son de dos *tipos*: libros y artículos”

Otra perspectiva la constituyen los *verbos* que desarrollan una *preferencia semántica* en el dominio que nos ocupa. Estos verbos son susceptibles de aparecer en cualquier texto descriptivo y, aunque pueden tener varios significados, sólo uno de ellos tiene preferencia en los textos que se tratan. Por ello, en este caso el análisis está guiado por la semántica. Por ejemplo, el verbo “disponer” puede denotar, entre otros, los siguientes significados:

1. DISPONER: Significado “mandar”

verbo de acción que requiere un Agente(Agt) y un Objeto (Obj):

esquema 1: Alguien (Agt) *dispone* Algo (Obj)

2. DISPONER: Significado “poseer”

verbo de estado que requiere un Beneficiario (Ben) y un Objeto (Obj):

esquema 1: Algo/Alguien (Ben) *dispone de* Algo/Alguien (Obj)

esquema 2: Se *dispone de* Algo/Alguien (Obj) en Algo (Ben)

esquema 3: Se *dispone de* Algo (Obj) *de* Algo (Ben)

De estos dos significados, el preferente en los textos descriptivos de bases de datos es el segundo (poseer).

Para los verbos que no tienen una preferencia semántica existe otra perspectiva lingüística que inicia un análisis sintáctico para buscar el análisis que mejor encaja en la oración. Un verbo de este tipo es “dirigir”, cuyos significados, entre otros, son:

1. DIRIGIR: significado “gobernar”

Verbo de acción que requiere un Agente(Agt) y un Objeto (Obj):

Alguien (Agt) *dirige* Algo (Obj)

2. DIRIGIR: significado “movimiento”

Verbo que requiere un Agente que sea a la vez Objeto (Agt=Obj) y un Locativo (Loc):

Alguien/Algo (Agt=Obj) *se dirige* a Algo (Loc)

También se presta atención a las interrelaciones que existen entre los componentes de un sintagma nominal o preposicional de igual importancia a las que existen entre un verbo y sus argumentos. Por ejemplo, en las dos oraciones siguientes existe un patrón sintáctico *nombre-adjetivo* que resulta de gran utilidad en la identificación de generalizaciones entre tipos de entidad:

“El personal académico imparte cursos y puede realizar trabajos de investigación”

“Los estudiantes graduados investigan”

Clase Semántica del Verbo	Roles Semánticos	Verbos de Estado		Representación Gráfica
		Pragmática del Verbo	Roles Pragmáticos	
Benefactivo		Descripción	Ben= Tipo de Entidad Obj= Atributo	
	Ben, Obj	Asociación Genérica	Ben= Tipo de Entidad Obj= Tipo de Entidad	
		Identificación	Ben= Tipo de Entidad Obj= Clave_Primary	
Locativo		Agregación	Loc=Colección Obj= Miembro	
	Loc, Obj	Asociación Genérica	Loc= Tipo de Entidad Obj= Tipo de Entidad	
		Generalización	Obj= Supertipo Loc= Subtipo	

Tabla 9.1. Correspondencias semántico-pragmáticas

Una vez se ha realizado el análisis lingüístico basado en las perspectivas lingüísticas, es necesario buscar la correspondencia de las estructuras semánticas obtenidas con los distintos constructores de un modelo conceptual de datos (conocimiento pragmático), así como resolver las posibles inconsistencias que pudieran surgir según se van analizando las oraciones (conocimiento del discurso). La Tabla 9.1 muestra algunas de las correspondencias semántico-pragmáticas de los verbos de estado benefactivos y locativos.

3. CARACTERÍSTICAS DEL ESQUEMA CONCEPTUAL

Como resultado de la fase de modelado conceptual se obtendrá un esquema conceptual que debe cumplir los siguientes objetivos:

- Captar y almacenar el universo del discurso mediante una descripción rigurosa, representando la información que describe a la organización y que es necesaria para su funcionamiento.
- Aislar la representación de la información de los requisitos de la máquina y exigencias de cada usuario particular.
- Independizar la definición de la información de los SGBD en concreto.

En resumen, como se dice en ANSI (1977), “un esquema conceptual comprende la descripción central única de los distintos contenidos de información que pueden existir en una base de datos”.

Para ello debemos contar con un buen modelo de datos. Nosotros proponemos el E/R con algunas extensiones, que permita obtener esquemas conceptuales que se caractericen por su:

- **Claridad**, esto es, que la significación no sea ambigua.
- **Coherencia**, es decir, que no existan contradicciones o confusiones.
- **Plenitud**, en cuanto a que el esquema conceptual ha de representar lo esencial del fenómeno sin buscar la exhaustividad.
- **Fidelidad**, en el sentido de que la representación del universo del discurso ha de hacerse sin desviaciones ni deformaciones.
- **Sencillez**, se ha de buscar la máxima sencillez sin atentar contra las anteriores características.

sencillez del esquema conceptual ha de estar basada en que:

- El número de componentes básicos debe ser tan reducido como sea posible.
- Ha de separar claramente conceptos distintos.
- Debe preservar la simetría, es decir, no destruir las simetrías naturales.
- La redundancia tiene que ser cuidadosamente controlada.

Hay que destacar, sin embargo, que la aplicación de las características anteriores siempre resulta fácil, puesto que a veces unas van en detrimento de otras. Además, debe ser consciente de que de un mismo universo del discurso se pueden obtener

distintos esquemas conceptuales, y se deberá decidir cuál es el que mejor cumple las cualidades que consideramos de mayor interés, además de tener la adecuada capacidad semántica.

El problema de la equivalencia de esquemas E/R es de gran interés, dado que es inevitable la subjetividad del diseñador. Lo importante es conseguir establecer en qué medida cada solución propuesta es capaz de recoger la semántica inmersa en nuestro universo del discurso, a fin de poder elegir aquel esquema que incorpore el mayor número de restricciones semánticas de nuestro mundo real. A este respecto hay que destacar el trabajo de JAJODIA *et al.* (1983b), en el que se establecen tres criterios de equivalencia entre diagramas E/R, que, de menos a más estrictos, son:

- *Compatibilidad de dominios de datos*, que asegura que los diagramas representan, en conjunto, el mismo UD.
- *Equivalencia de dependencias de datos*, que asegura que los diagramas satisfacen las mismas restricciones (dependencias funcionales embebidas en el diagrama) entre los datos representados.
- *Equivalencia de ejemplares*, que requiere que los esquemas E/R puedan almacenar los mismos ejemplares de nuestro universo del discurso.

4. METODOLOGÍAS ASCENDENTES Y DESCENDENTES

Tradicionalmente, en el diseño de las bases de datos se han venido utilizando distintas metodologías para elaborar el esquema conceptual, apoyadas en distintos modelos conceptuales (modelos E/R, modelo relacional extendido (RM/T), modelo infológico, modelo semántico en red, modelo binario, etc.)²

En general las distintas metodologías pueden agruparse en:

- *descendentes (top-down)*: cuya filosofía responde a que “el esquema conceptual refleje directamente la visión de la empresa que se intenta modelar en la BD”. Se parte del estudio del universo del discurso para elaborar el esquema conceptual y, posteriormente, sobre él se definen las vistas de usuario como subconjuntos de este esquema conceptual (véase Figura 9.8).

² En estos momentos el MER con distintas extensiones es el que se ha impuesto en las distintas metodologías de diseño.

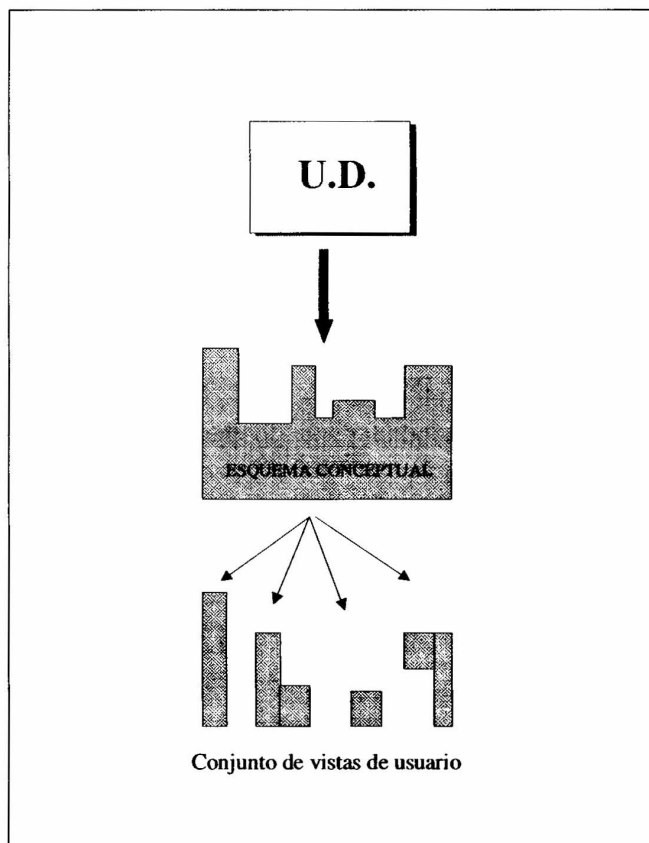


Figura 9.8. Diseño descendente

Este tipo de metodologías supone que los diseñadores conocen bien los requisitos del sistema de información, por lo que pueden identificar las entidades existentes en el mismo con sus atributos, para pasar a definir las distintas interrelaciones entre estas entidades y los atributos propios de ellas.

Una vez elaborada la estructura, se pueden establecer las vistas parciales para comprobar que cumplen los requisitos locales y, en caso contrario “retocar” el esquema hasta conseguir atender las necesidades de los usuarios.

- *ascendentes (bottom-up)*: este tipo de metodologías entiende el esquema conceptual como “el resultado de la integración de las vistas de los grupos de usuarios” —subsistemas—, por lo que se empieza construyendo las vistas de cada uno de estos subsistemas (que corresponde a las aplicaciones más importantes) y, teniendo en cuenta las restricciones entre dichas vistas, se

elabora el esquema conceptual mediante un proceso de integración de vistas, tal como se muestra en la figura 9.9.

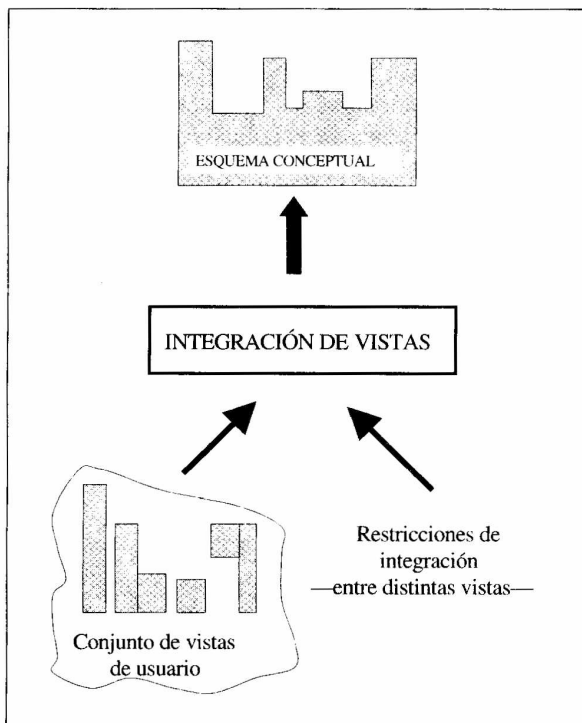


Figura 9.9. Diseño ascendente

Nosotros proponemos, en especial cuando se trata de grandes sistemas muy complejos, integrar ambos enfoques, iterando la elaboración del esquema conceptual descendente y ascendentemente hasta que se logren cumplir los requisitos de información deseados.

De todos modos, la aplicación de una u otra metodología (ascendente o descendente) dependerá mucho de la complejidad y tamaño de la BD.

En una base de datos compleja y de gran tamaño se empezará diseñando los esquemas de los distintos subsistemas, siguiendo para cada uno de ellos una metodología descendente; esto es, para cada subsistema se elaboraría su correspondiente esquema E/R con sus entidades, atributos e interrelaciones, derivando las vistas parciales a fin de comprobar que se cumplen los requisitos y, si no fuese así, se iría *retocando* el esquema. Posteriormente, se produciría la integración de todos los esquemas de estos subsistemas, elaborando de manera ascendente el esquema global, y

comprobando este esquema descendientemente para ver si refleja la concepción del mundo real relativo a la empresa en su conjunto; luego puede comprobarse ascendientemente este esquema y así ir refinándolo sucesivamente.

Cuando se trata de una base de datos menos compleja y/o de un tamaño medio o pequeño, no sería necesario el proceso de integración de vistas, ya que desde un principio se podría elaborar el esquema global de todo el sistema.

El proceso de integración de esquemas o *vistas de subsistemas* (que algunos autores llaman, en nuestra opinión de forma poco precisa; *superposición de modelos*) reviste gran interés, por lo que pasaremos a expresarlo con más detalle.

5. EL PROCESO DE “INTEGRACIÓN DE VISTAS”

Las posibilidades que pueden darse en la integración de vistas se resumen en la figura 9.10 que presenta la taxonomía propuesta por NAVATHE y GADGIL (1982). En esta clasificación se distinguen, por un lado, las *vistas idénticas*, que son aquellas en las que se encuentran los mismos tipos de objetos, aunque puede que con distintos nombres; por otro lado están las *vistas no idénticas*, denominadas así por poseer (en todo o en parte) distintos tipos de objetos. Dentro de estas últimas, hay que distinguir aquellas que, aunque no sean idénticas, sí resultan equivalentes (por ejemplo, porque o que en una vista es un atributo en la otra está representado por una entidad) de las que no son equivalentes.

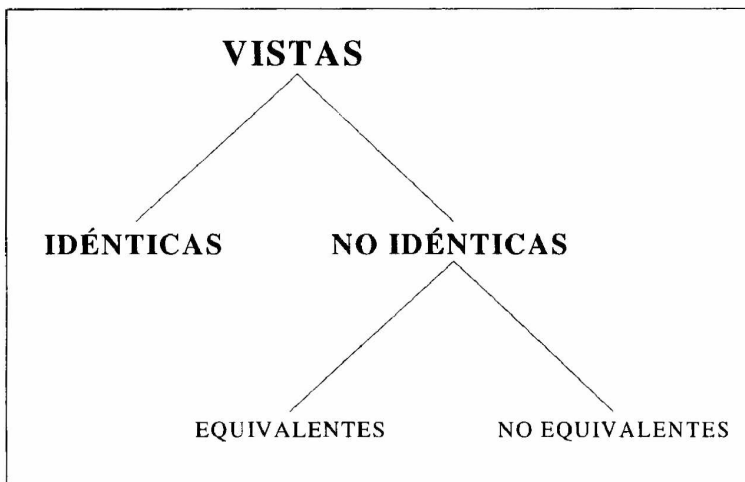


Figura 9.10. Clasificación de las vistas en el proceso de integración

La integración de vistas consiste en partir de dos vistas y obtener una nueva vista que las englobe, con ésta y una tercera se obtiene una nueva vista, y así sucesivamente, hasta llegar al esquema global que refleje la estructura de información de la empresa.

Otra forma de considerar el proceso de integración de vistas se muestra en la figura 9.11, donde del análisis del universo del discurso se elabora un esquema conceptual preliminar o *bruto* (también se puede partir del esquema del sistema anterior si éste existe), así como una serie de vistas complementarias; con estas vistas y el esquema conceptual bruto se inicia el proceso de integración, resultando al final del proceso el esquema conceptual definitivo de la base de datos que es denominado por algunos autores [YAO (1985)] *modelo de información global*. Actuando de esta manera no se integran distintas vistas, de dos en dos para elaborar el esquema conceptual, sino que, en realidad, se está utilizando el proceso de integración de vistas para *refinar* el esquema conceptual. Ambas posibilidades son admitidas dentro del marco de nuestra metodología.

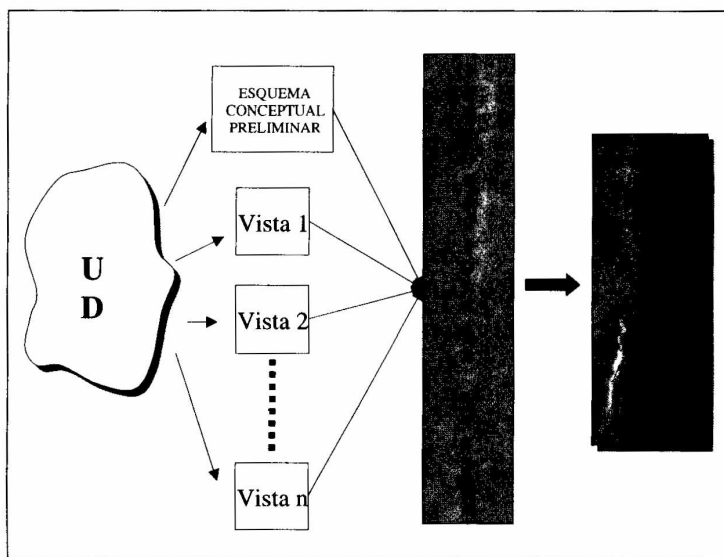


Figura 9.11. Proceso de integración de vistas

En el proceso de integración de vistas distinguimos dos etapas:

5.1. Resolución de conflictos

Al querer integrar distintas vistas se pueden producir varios problemas:

A) Conflictos de nombres

Pueden ser tanto *homonimia* (a dos objetos distintos se les ha asignado el mismo nombre) como de *sinonimia* (un mismo objeto que posee más de un nombre). Este tipo de conflictos se resuelve de forma cómoda mediante la ayuda de un diccionario de datos. Algunos autores, como BISKUP (1986), denominan a este tipo de conflicto *restricción de identidad*.

La solución es evidente, cambiar el nombre o los nombres a los objetos. Veamos algunos ejemplos.

- 1) *Conflicto de nombres en entidades*. Supongamos, en nuestro ejemplo de la universidad, que se tiene una entidad denominada PROFESOR con un atributo *Cód_Profesor* que sirve para identificarla y por otro lado (es decir, en una vista de otro subsistema) la entidad INSTRUCTOR con un código identificador *Cód_Instructor*. Al integrar estas dos vistas y analizar su contenido se podría llegar a la conclusión de que se trata de la misma entidad y del mismo atributo. La solución de este conflicto entre las dos vistas sería fácil: adoptar uno de los dos nombres (o uno nuevo) para designar a esa entidad en la vista resultante de la integración (por ejemplo, PROFESOR y *Cód_Profesor*).
- 2) *Conflicto de nombres en interrelaciones*. El conflicto de nombres puede darse también entre interrelaciones (véase la figura 9.12) y la solución siempre consiste en el cambio de nombre(s).

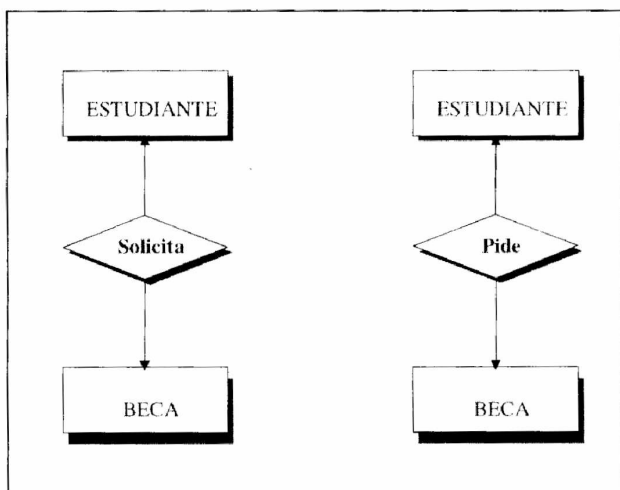


Figura 9.12. Conflicto de nombres en interrelaciones

B) Conflicto entre entidades

Pueden ser de varios tipos, los más comunes se producen cuando:

- 1) *Una entidad es un subconjunto de otra.* En este caso la solución consiste en introducir un subtipo. Por ejemplo, el conflicto entre las entidades REVISTA y PUBLICACIÓN (que incluye, además de revistas, recopilaciones y otro tipo de obras) se puede resolver introduciendo la revista como un subtipo de publicación. Es lo que algunos autores, como BISKUP (1986), llaman *restricción de selección*.
- 2) *Una entidad es disjunta con respecto a otra, pero ambas poseen atributos comunes; es decir, son un subtipo de una tercera entidad.* La solución es crear esa tercera entidad, esto es, el *supertipo*. Para seguir utilizando el ejemplo de la universidad, supóngase que tenemos estudiantes y profesores, existen determinados atributos comunes a ambos (como nombre y apellidos, dirección, DNI, etc.), pero existen también atributos propios de los estudiantes (como tema de doctorado, institución en que se desarrolla el trabajo, etc.) y otros propios de los profesores (como tipo de profesor, materia que imparte, etc.). Para integrar ambas entidades se puede crear una superentidad denominada PERSONA, que contendrá los atributos comunes, mientras que en las entidades ESTUDIANTE y PROFESOR estarán los atributos propios de cada una de ellas; solucionándose de esta manera el conflicto de entidades. Es la denominada por BISKUP (1986) *restricción de disyunción*.

C) Conflicto entre tipos de objetos en los que un atributo en una vista es una entidad en otra, o viceversa

La solución es transformar el atributo en entidad o la entidad en atributo, según convenga. Así, por ejemplo, si existen atributos de este *atributo*, o si éste está interrelacionado con otras entidades, convendrá considerarlo como entidad.

Veamos un caso concreto en el ejemplo de la universidad. La entidad PROFESOR puede poseer como atributo el nombre de la materia que imparte, y sin embargo en otra vista puede considerarse a la materia como entidad (véase Figura 9.13). Si pensamos que es importante almacenar ciertas propiedades de la materia además de su nombre, como pueden ser el número de horas teóricas, número de horas prácticas, etc., resultaría más conveniente considerarlo como entidad.

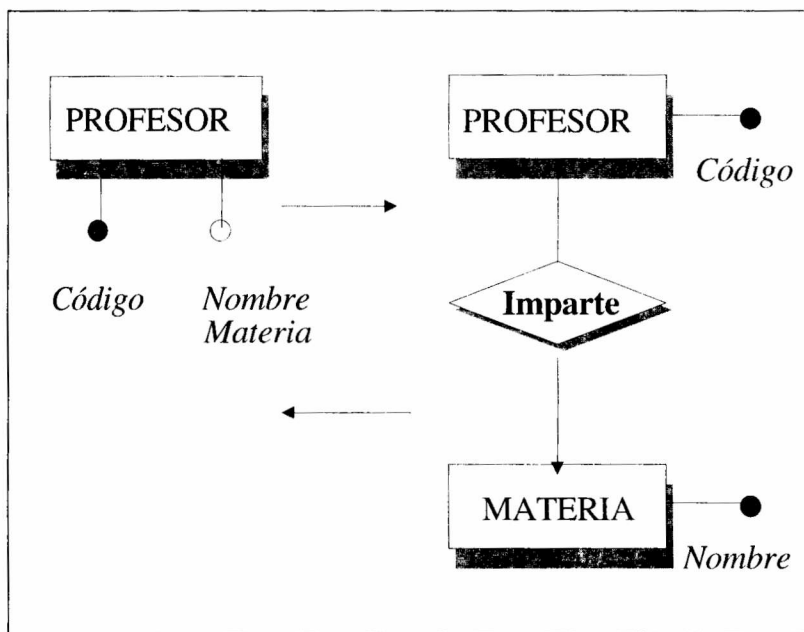


Figura 9.13. Conflicto entre tipos de objetos

D) Conflictos de cardinalidades en interrelaciones

Pueden reflejar que las dos interrelaciones son la misma, que hay dos interrelaciones distintas o que una de las entidades involucradas en la interrelación tiene uno o varios subtipos.

En la Figura 9.14 se muestran dos vistas distintas, una en la que la interrelación entre DOCTOR y EDICIÓN es de tipo 1:N y otra en que es de tipo N:M, puede suceder que:

- i.) Se trate de la misma interrelación, como **Imparte**; entonces se dejarían las cardinalidades menos restrictivas de ambas vistas, en nuestro caso se consideraría la interrelación de tipo N:M.
- ii.) Se trate de dos interrelaciones distintas, como **Imparte** (de tipo N:M) y **Dirige** (de tipo 1:N, suponiendo que una edición de un curso de doctorado puede ser dirigido por un doctor). En este caso debemos reflejar ambas interrelaciones con distintos nombres. Este ejemplo nos muestra la importancia que puede llegar a tener la asignación de nombres a los objetos de la base de datos.

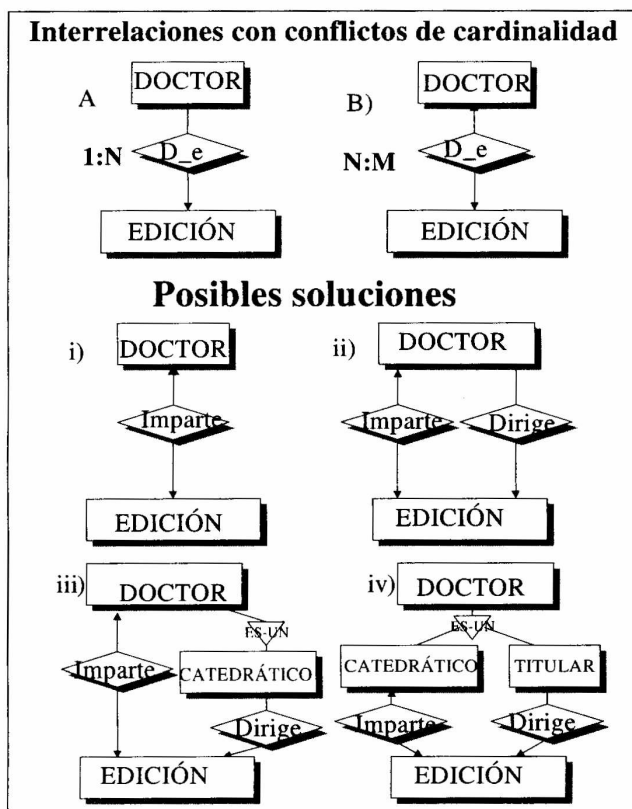


Figura 9.14. Conflicto entre cardinalidades de una interrelación

- iii.) La entidad DOCTOR tiene una interrelación con EDICIÓN que es **Imparte**, mientras que un subtipo de ella (por ejemplo, CATEDRÁTICO) tiene otra interrelación con EDICIÓN que es **Dirige**.
- iv.) Por último, y como variante del anterior, existen dos subtipos de la entidad DOCTOR que poseen interrelaciones distintas con la entidad EDICIÓN, por ejemplo, el subtipo TITULAR y el subtipo CATEDRÁTICO con las interrelaciones **Imparte** y **Dirige**, respectivamente.

5.2. Análisis de redundancias de interrelaciones

Una vez integradas las vistas, habrá que analizar si se producen redundancias de interrelaciones, lo que puede reflejarse gráficamente como ciclos en el diagrama E/R. Estos ciclos se deben detectar y estudiar tal como hemos visto en el capítulo 3.

Como podemos observar, la integración de vistas es una fase muy importante en el diseño de bases de datos, que puede resultar bastante complicada en caso de no realizarse con el soporte de una herramienta interactiva (con un potente diccionario) que ayude al diseñador a lo largo del proceso.

En NAVATHE *et al.* (1986) se muestra el esquema de un sistema para integración de vistas (véase figura 9.15) en el que el diseñador interactúa con el sistema especificando las restricciones *intervistas e intravistas* y la política de integración o la prioridad entre las distintas vistas. El sistema gestiona mediante un diccionario de datos no sólo estas entradas al proceso, sino también las salidas: la correspondencia entre las vistas y el esquema integrado, así como los conflictos entre las vistas.

También se han investigado y construido herramientas CASE y sistemas expertos que automatizan en parte el proceso de integración de vistas, ayudando también en toda la etapa de diseño conceptual (véanse, por ejemplo, BOUZEGHOUB, GARDARIN y METAIS (1985), SHETH *et al.* (1988), DOGAC, YURUTEN y SPACCAPIETRA (1989) o HAYNE y RAM (1990).

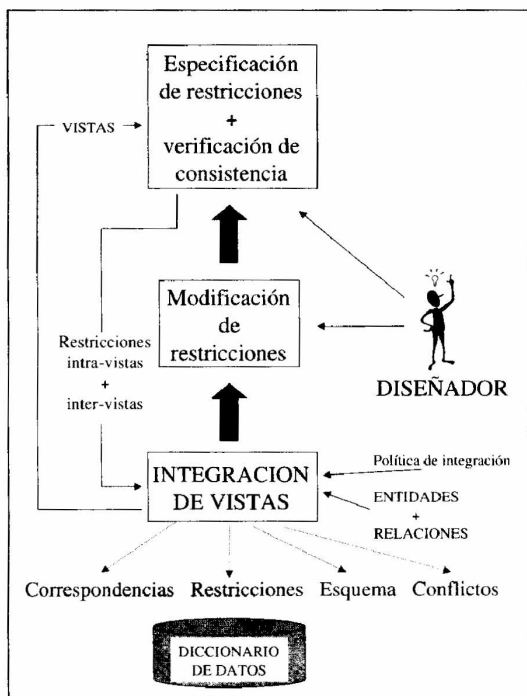


Figura 9.15. Esquema de un sistema para integración de vistas
—NAVATHE *et al.* (1986)—

DISEÑO LÓGICO ESTÁNDAR

En este capítulo abordaremos la etapa de diseño lógico estándar, que está basada en el modelo relacional. Empezaremos con una visión global de la fase de diseño lógico para centrarnos en las reglas de transformación del esquema E/R al relacional estándar, poniendo ejemplos de los distintos casos que se pueden dar, así como su especificación en el SQL estándar. Finalizaremos exponiendo una herramienta muy útil que sirve para visualizar bases de datos relacionales: el *grafo relacional*, también denominado por algunos autores, *grafo de combinación*.

1. ETAPAS DEL DISEÑO LÓGICO

En el diseño lógico se deben coordinar exigencias casi siempre encontradas, como son eliminar redundancias, conseguir la máxima simplicidad y evitar cargas suplementarias de programación, obteniendo una estructura lógica adecuada que venga a establecer el debido equilibrio entre las exigencias de los usuarios y la eficiencia.

Habrà que conseguir, por tanto, equilibrar los distintos requisitos exigidos al sistema: flexibilidad, confidencialidad, integridad, tiempo de respuesta, etc., estableciendo unas prioridades y adoptando una solución de compromiso.

En la metodología que proponemos, introducimos las características del SGBD lo más tarde posible, ya que, a nuestro juicio, todo producto informático (y una base de datos, es uno más) debe poseer, como ya se ha señalado, la *portabilidad* como una de sus características más deseables.

Esta portabilidad es necesaria para desarrollar productos que puedan ser implementados sobre distintos SGBD, y para facilitar la migración entre versiones de un mismo sistema. Este problema de las migraciones está siempre de actualidad en los SGBDR debido a la continua mejora de los mismos, así como a la implementación de un número cada vez mayor de conceptos del modelo que no se encontraban, (o todavía no se encuentran) en los productos comerciales, como integridad referencial, dominios, etc.

Las etapas que proponemos dentro del diseño lógico son las siguientes:

A) Diseño lógico estándar

A partir del esquema conceptual resultante de la etapa anterior, y teniendo en cuenta los requisitos de proceso y de entorno, se elabora un esquema lógico estándar (ELS), que se apoya en un modelo lógico estándar (MLS), el cual será el mismo modelo de datos (Jerárquico, Codasyl o Relacional) soportado por el SGBD que se vaya a utilizar, pero sin las restricciones ligadas a ningún producto comercial. Por tanto, al llegar a esta etapa será preciso haber realizado ya la selección del sistema o, al menos, haber decidido el modelo de datos con el que se va a trabajar. En nuestro caso el MLS es el modelo relacional, pero, como ya hemos señalado, la metodología se podría aplicar igualmente a los modelos Jerárquico o Codasyl.

Este ELS se describirá utilizando el lenguaje estándar, si existe, del modelo de datos correspondiente (SQL, NDL, etc.). En nuestra metodología se usa el SQL2, ISO (1992).

B) Diseño lógico específico

Con el ELS, y teniendo en cuenta el modelo lógico específico (MLE) propio del SGBD (INGRES, SYBASE, DB2, ORACLE, INFORMIX, INTERBASE, etc.), se elabora el esquema lógico específico (ELE), que será descrito en el lenguaje de definición de datos (LDD) del producto comercial que estemos utilizando.

En la fase de diseño lógico, además de los MLS, MLE y los lenguajes SQL estándar y el SQL propio del SGBD utilizado, disponemos de otras herramientas, como son los diagramas de dependencias funcionales, la teoría de la normalización, los grafos relacionales, etc. El diseño lógico estándar, así como sus herramientas, serán objeto de los siguientes epígrafes; mientras que el diseño lógico específico se estudiará junto con el diseño físico en el próximo capítulo, ya que ambos son muy dependientes del SGBD en el que se implementará la base de datos.

Un resumen de las etapas de la fase de diseño lógico se encuentra en la figura 10.1.

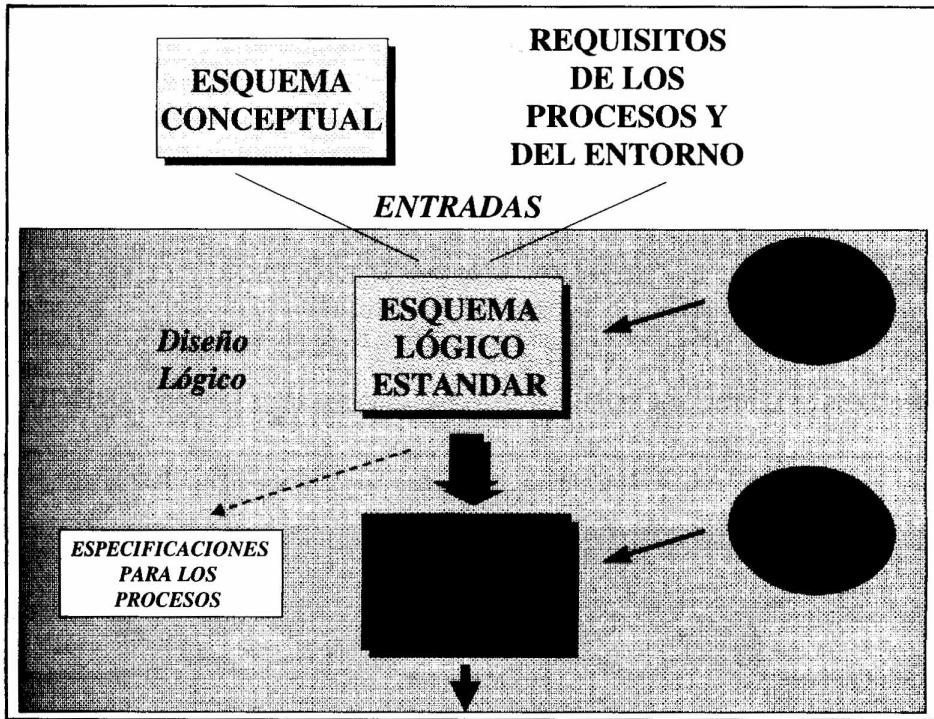


Figura 10.1. Entradas y etapas del diseño lógico

2. TRANSFORMACIÓN DEL ESQUEMA CONCEPTUAL AL LÓGICO ESTÁNDAR

Las tres reglas básicas para convertir un esquema en el modelo E/R al relacional son las siguientes:

- 1) Todo tipo de entidad se convierte en una relación.
- 2) Todo tipo de interrelación N:M se transforma en una relación.
- 3) Para todo tipo de interrelación 1:N se realiza lo que se denomina *propagación de clave* (regla general), o bien se crea una nueva relación.

Debido a que el modelo relacional no distingue entre entidades e interrelaciones, ambos conceptos deben representarse mediante relaciones. Esto implica una pérdida de semántica con respecto al esquema E/R, ya que las interrelaciones N:M no se

distinguen de las entidades y las 1:N se representan mediante una propagación de clave, desapareciendo incluso el nombre de la interrelación.

En el ejemplo de la figura 10.2 puede observarse que las tres entidades DEPARTAMENTO, PROFESOR y CURSO se han transformado en otras tantas relaciones. La interrelación N:M **imparte** da lugar a una nueva relación cuya clave es la concatenación de las claves primarias de las entidades que participan en ella (*Cód_prof* de PROFESOR y *Cód_curso* de CURSO), siendo además estas claves ajenas de **imparte**, que referencian a las tablas PROFESOR y CURSO, respectivamente; la interrelación 1:N **pertenece** se ha transformado mediante el mecanismo de propagación de clave, por el que se ha incluido en la tabla PROFESOR el atributo clave de la entidad DEPARTAMENTO (*Nombre_dep*), que constituye, por tanto, clave ajena de la relación PROFESOR referenciando a la tabla DEPARTAMENTO.

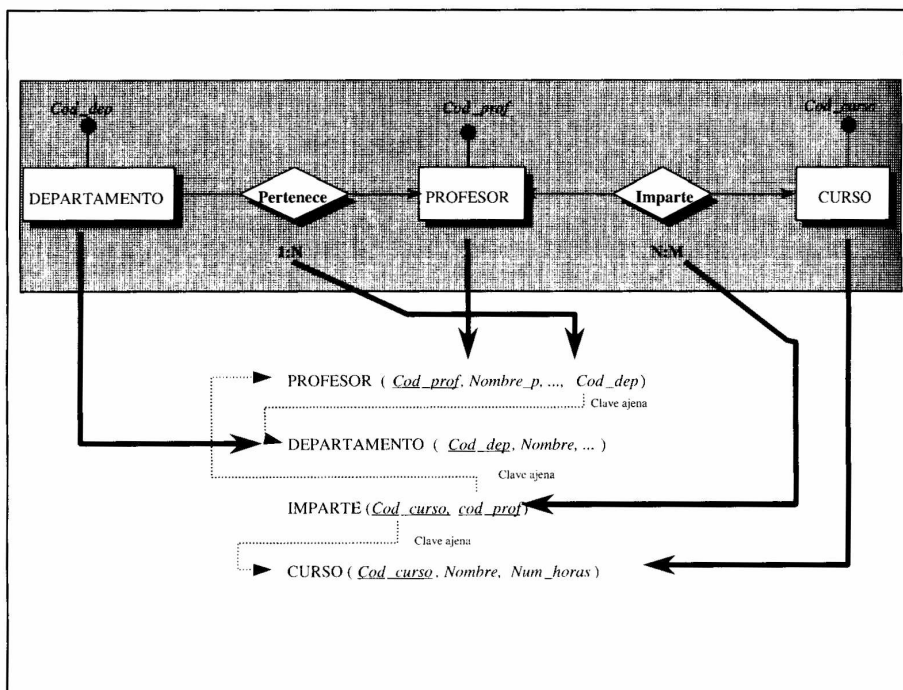


Figura 10.2. Ejemplo de paso del ME/R al modelo relacional

Teniendo en cuenta que el modelo E/R básico tiene otros objetos y que, además, hemos propuesto una serie de extensiones, hemos de ver cómo se puede recoger el modelo E/R completo (con sus extensiones) en el modelo relacional y señalar, en su caso, aquellas características que, debido a la menor semántica del modelo relacional, no es posible representar directamente, a fin de implementarlas posteriormente por

edio de disparadores, procedimientos almacenados o procedimientos de usuario, con problemas que en este último caso pueden aparecer relativos al mantenimiento de integridad del sistema.

REGLAS CONCERNIENTES AL MODELO BÁSICO

- 1. Transformación de dominios.** En el modelo relacional estándar un dominio es un objeto más, propio de la estructura del modelo que, como tal, tendrá su definición concreta en el LDD (en nuestro caso el SQL92) que se elija. Como ejemplo podemos crear el dominio de los estados civiles, que es un conjunto de valores de tipo carácter, de longitud 1, que puede tomar los valores 'S', 'C', 'V' o 'D' (véase figura 10.3).

En el SQL propuesto expresaríamos este dominio de la siguiente forma:

```
CREATE DOMAIN Estados_Civiles AS CHAR(1)
CHECK (VALUE IN ('S', 'C', 'V', 'D'))
```

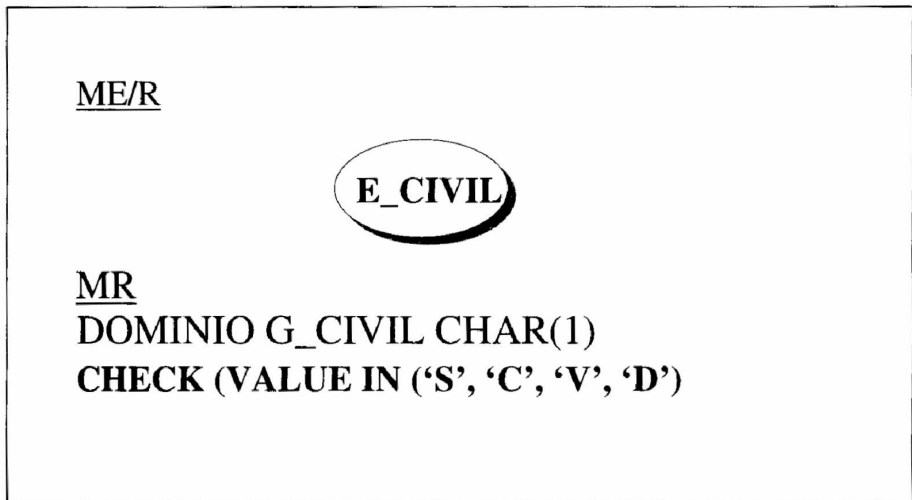


Figura 10.3. Transformación de dominios

Observación: No debemos olvidar, e insistimos en ello, que el modelo lógico estándar, admite dominios, aunque no existan en la mayoría de los productos comerciales. Será en la transformación del MLS al MLE cuando habremos de buscar solución al problema, tratando de evitar la pérdida de semántica que origina la pobreza de muchas implementaciones del modelo relacional.

2. Transformación de entidades. Según lo que hemos indicado en la introducción de este capítulo, "cada tipo de entidad se convierte en una relación". Esto es, el modelo lógico estándar posee el objeto RELACION o TABLA mediante el cual representamos las entidades. La tabla se llamará igual que el tipo de entidad de donde proviene. Para su definición disponemos en el SQL de la sentencia CREATE TABLE. Por ejemplo, la entidad PROFESOR se transforma en una tabla con ese mismo nombre (véase figura 10.4). En este caso la transformación es directa, y no hay pérdida de semántica.

3. Transformación de atributos de entidades. Cada atributo de una entidad se transforma en una columna de la relación a la que ha dado lugar la entidad. Pero teniendo en cuenta que tenemos atributos identificador principal, otros que son identificadores alternativos y el resto de atributos que no son identificadores —atributos no principales— desglosamos esta regla en tres subreglas:

3.1. Atributos identificadores. El (o los) atributo(s) que son identificador(es) principales (AIP en adelante) pasan a ser la clave primaria de la relación. Por ejemplo, en la figura 10.4 tenemos la relación PROFESOR, fruto de la transformación de la entidad del mismo nombre, con su AIP (*Cód_prof*) que pasa a ser la clave primaria.

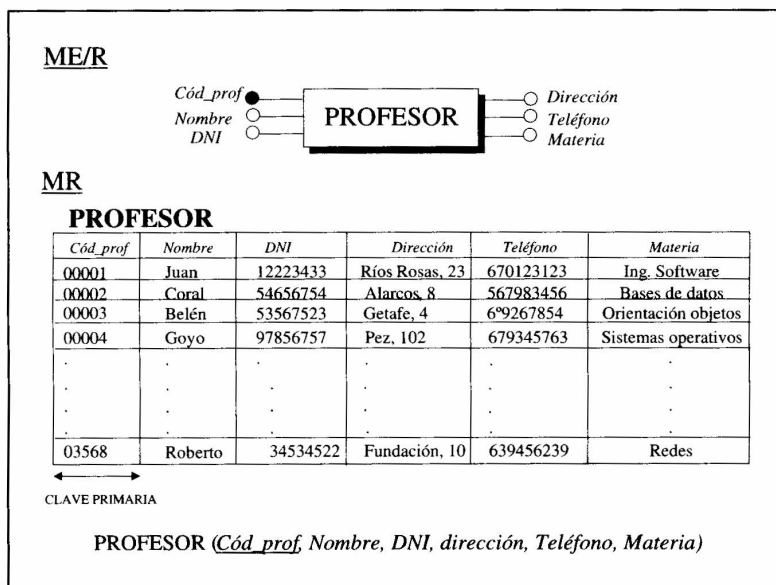


Figura 10.4. Transformación de una entidad

El lenguaje lógico estándar (LLS) recoge directamente este concepto por medio de la cláusula **PRIMARY KEY** en la descripción de la tabla, luego la transformación es directa y no hay pérdida de semántica.

- 3.2. *Atributos identificadores alternativos.* Respecto a los atributos identificadores alternativos el MLS recoge por medio de la cláusula **UNIQUE** estos objetos, Ya que son soportados directamente por el modelo relacional. Al ser la transformación directa, no hay pérdida de semántica. Si se desea que estos atributos no tomen valores nulos habrá que indicarlo.
- 3.3. *Atributos no identificadores.* Estos atributos pasan a ser columnas, como los anteriores, de la relación, las cuales tienen permitido tomar valores nulos a no ser que se indique lo contrario.

Aplicando las reglas 2 y 3, la transformación de la entidad **PROFESOR** al modelo relacional estándar se representa mediante el LLS.

```
CREATE TABLE Profesor(  
    Cód_Profesor Códigos,  
    Nombre Nombres,  
    DNI DNIS NOT NULL  
    Dirección Lugares,  
    Teléfono Nos_Teléfono,  
    Materia Materias,  
    PRIMARY KEY (Cód_Profesor),  
    UNIQUE (DNI));
```

4. **Transformación de interrelaciones.** Ya hemos señalado que, dependiendo del tipo de correspondencia de la interrelación, y de otros aspectos semánticos de la misma variará la manera de realizar la transformación al esquema relacional, por eso desglosamos esta regla en tres subreglas:

- 4.1. *Interrelaciones N:M.* Un tipo de interrelación N:M se transforma en una relación que tendrá como clave primaria la concatenación de los AIP de los tipos de entidad que asocia. Se ve claramente, como ya hemos indicado anteriormente, que dentro de un esquema relacional no hay manera de diferenciar qué relaciones provienen de una entidad y cuáles de ellas proceden de la transformación de interrelaciones, por lo tanto hay cierta pérdida de semántica en este punto de la transformación; semántica que sólo puede ser salvada, almacenando comentarios sobre la procedencia de cada una de las tablas o mediante un convenio en la denominación de estas tablas que provienen de interrelaciones en el modelo E/R.

Por ejemplo, la figura 10.5 muestra esta transformación, en la que se presenta la asociación que existe entre los profesores y los cursos que imparten, apareciendo una relación cuya clave primaria está compuesta por la concatenación del código del profesor y el código del curso.

Además, cada uno de los atributos que forman la clava primaria de esta relación son claves ajenas que referencian a las tablas en que se ha convertido las entidades interrelacionadas (claves primarias), lo que se especifica en el LLS a través de la cláusula FOREIGN KEY dentro de la sentencia de creación de la tabla. Habrá que estudiar, además, qué ocurre en los casos de borrado y modificación de la clave primaria referenciada, teniendo en cuenta que en nuestro LLS las opciones permitidas son operación restringida (en caso de no especificar la acción o poner NO ACTION), puesta a nulo (SET NULL), puesta a valor por defecto (SET DEFAULT) u operación en cascada (CASCADE).

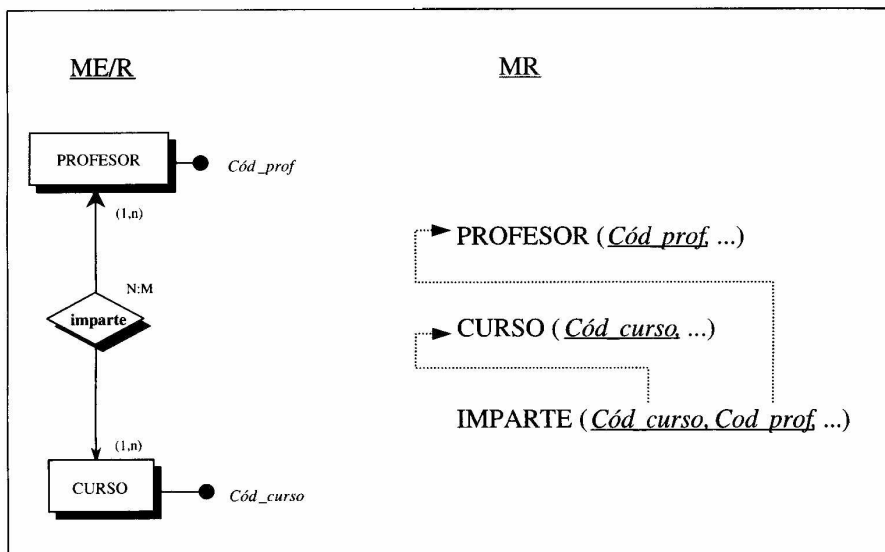


Figura 10.5. Transformación de una interrelación N:M

Aplicando lo que acabamos de decir al ejemplo de la figura 10.5, obtendríamos en SQL:

```
CREATE TABLE Imparte
(Cód_Profesor Codigos_P,
Cód_Curso Codigos_C,
....,
PRIMARY KEY (Cód_Profesor, Cód_Curso),
```



```

FOREIGN KEY (Cód_Profesor) REFERENCES Profesor
ON DELETE CASCADE
ON UPDATE CASCADE,
FOREIGN KEY (Cód_Curso) REFERENCES Curso
ON DELETE CASCADE
ON UPDATE CASCADE)

```

Sería también correcto no dar las opciones de borrado y modificación (o lo que es lo mismo, poner NO ACTION), en cuyo caso se rechazarían el borrado o modificación de aquellas tuplas de las tablas referenciadas cuando su valor de la clave primaria existiese en la tabla que referencia. En cambio, no se admitirían las opciones de puesta a nulos o a valor por defecto.

Otra característica que debemos recoger en esta transformación es la cardinalidad mínima de cada una de las entidades que participan en la interrelación, lo que se hace mediante la especificación de restricciones, aserciones o disparadores.

Un ejemplo de este caso sería aquel en el que suponemos que cada curso puede ser impartido por menos de 4 profesores, que podría quedar reflejado en la siguiente aserción:

```

CREATE ASSERTION Profesor_Curso
CHECK NOT EXIST (SELECT COUNT(*)
FROM IMPARTE
GROUP BY COD_CURSO
HAVING COUNT(*) ≥ 4)

```

4.2. *Interrelaciones 1:N.* Existen dos soluciones para la transformación de una interrelación 1:N:

- a) Propagar los AIP del tipo de entidad que tiene de cardinalidad máxima 1 a la que tiene N, es decir en el sentido de la flecha, desapareciendo el nombre de la interrelación, con lo cual se pierde semántica (ésta es la regla habitual). Podemos ver un ejemplo en la figura 10.6.
- b) Transformarlo en una relación, como si se tratara de una interrelación N:M; sin embargo en este caso, la clave primaria de la relación creada es sólo la clave primaria de la tabla a la que le corresponde la cardinalidad N.

Aunque depende del criterio del diseñador, e influye además de la semántica la eficiencia, los casos en los que puede ser apropiado transformar la interrelación en una relación son los siguientes:

- 1) Cuando el número de ejemplares interrelacionados de la entidad que propaga su clave es muy pequeño y, por tanto, existirían muchos valores nulos en la clave propagada. Por ejemplo, en la figura 10.7, en principio, existirían dos soluciones, pero si suponemos que el número de subtemas que pertenecen a un tema es muy pequeño en comparación con los que son independientes, puede no ser conveniente propagar la clave de tema a los que dependen de él, ya que aparecerían muchos valores nulos; por tanto, la solución b) puede ser adecuada¹ (por el solo hecho de existir algún ejemplar no interrelacionado la cardinalidad mínima es cero).

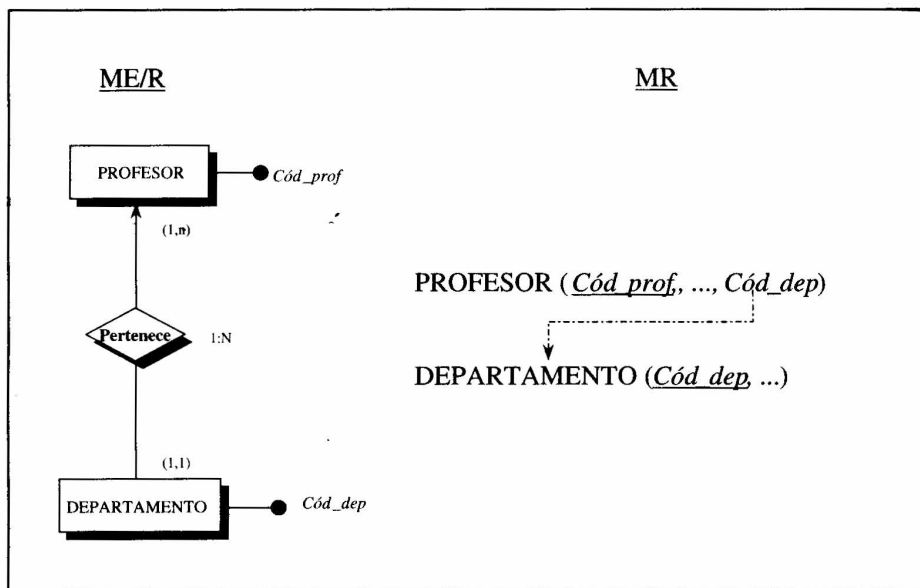


Figura 10.6. Transformación de una interrelación 1:N en propagación de clave

- 2) Cuando se prevé que dicha interrelación en un futuro se convertirá en una de tipo N:M.

¹ Hay que tener en cuenta que, con esta solución, la eficiencia en las consultas es menor, ya que si se desea recuperar los datos de un tema con el código del tema de nivel superior sería preciso hacer una combinación ("join") de las tablas (TEMA y CONSTA).

- 3) Cuando la interrelación tiene atributos propios y no deseamos propagarlos (a fin de conservar la semántica).

Por otro lado, la propagación de clave causa la aparición de claves ajenas, con sus mecanismos de borrado y actualización correspondientes, según la semántica del problema.

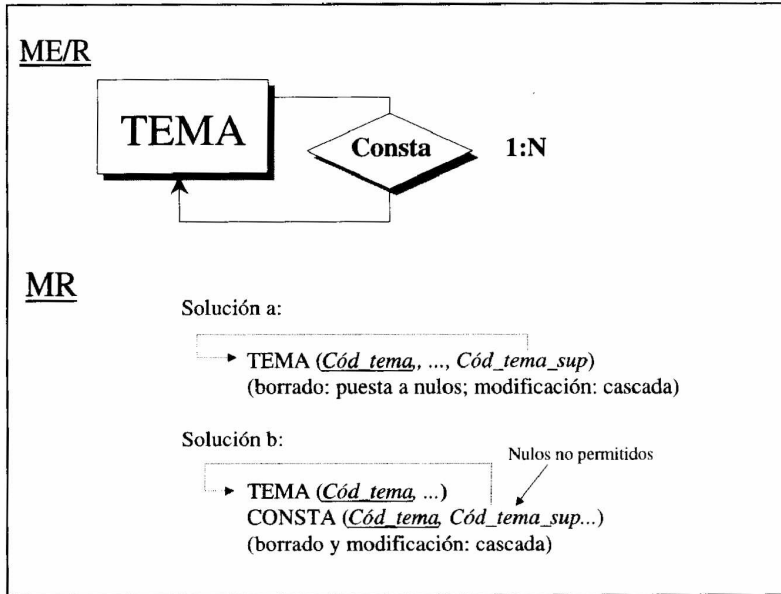


Figura 10.7. Transformación de una interrelación 1:N en una relación

Si transformamos la interrelación de la figura 10.7 en una relación (solución b), las correspondientes sentencias en SQL serían:

```
CREATE TABLE Consta (
  Cód_Tema Codigos,
  Cód_Tema Sup Codigos,
  PRIMARY KEY (Cód_tema)
  FOREIGN KEY (Cód_Tema) REFERENCES Tema
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (Cod_Tema_SUP) REFERENCES Tema
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

Si se transforma la interrelación en una relación, el control de las reglas de borrado y modificación se realiza de forma análoga a la de la regla 4. 1. Si se utiliza el mecanismo de propagación de clave, la cardinalidad mínima de la entidad para la cual la cardinalidad máxima es uno (entidad de nivel superior) se puede controlar impidiendo o permitiendo la existencia de nulos en la clave ajena propagada.

Como se puede observar en la figura 10.8, al ser la cardinalidad de DEPARTAMENTO (1,1), no pueden admitirse valores nulos en la clave propagada; sin embargo sí habrían de admitirse si la cardinalidad fuera (0,1).

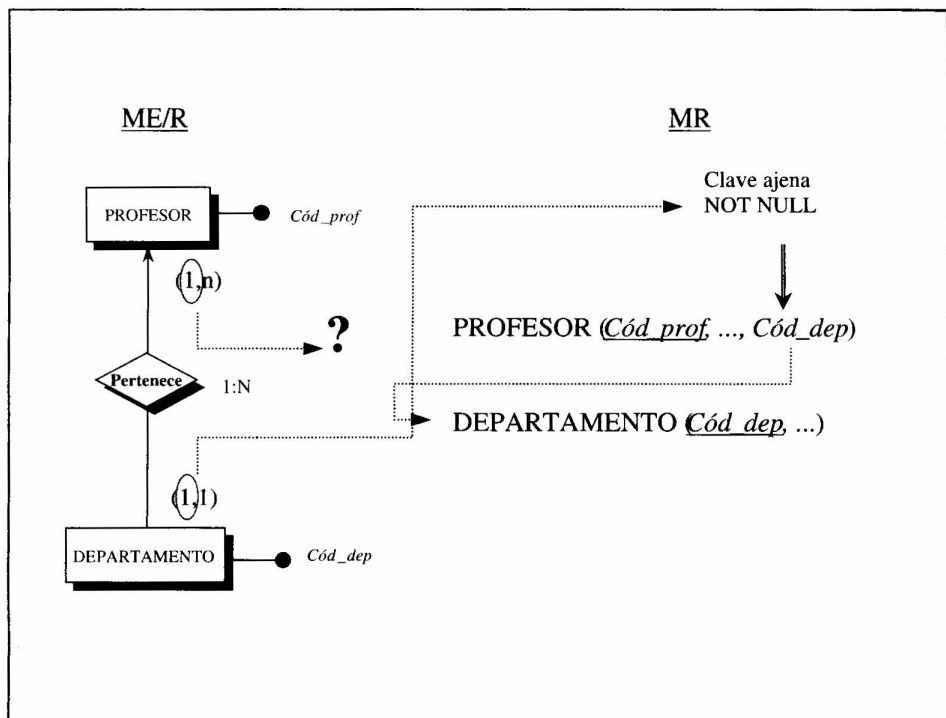


Figura 10.8. Transformación de cardinalidades mínimas

La cláusula NOT NULL no resuelve el problema de la cardinalidad mínima 1 en la entidad en la que se incluye la clave propagada, debiéndose definir para controlarla a la correspondiente restricción (check, aserción o disparador).

4.3. Interrelaciones 1:1. Una interrelación de tipo 1:1 es un caso particular de una N:M o, también, de una 1:N, por lo que no hay regla fija para la transformación de este tipo de interrelación al modelo relacional estándar, pudiéndose aplicar la regla 4.1 (con lo que crearíamos una relación) o aplicar la regla 4.2. (esto es, propagar la clave correspondiente). En este último caso hay que observar que en una interrelación 1:1, la propagación de la clave puede efectuarse en ambos sentidos.

Los criterios para aplicar una u otra regla y para propagar la clave se basan en las cardinalidades mínimas, en recoger la mayor cantidad de semántica posible, evitar los valores nulos o en motivos de eficiencia. A continuación exponemos algunos ejemplos que puedan servir de pauta al lector:

- a) Si las entidades que se asocian poseen cardinalidades (0,1), puede ser conveniente transformar la interrelación 1:1 en una relación. Por ejemplo, en la figura 10.9 tenemos la interrelación **Matrimonio** entre tipos de entidad **HOMBRE** y **MUJER**, la cual se transformará en una relación, evitando así los valores nulos que aparecerían en caso de propagar la clave de **MUJER** a la tabla **HOMBRE** o viceversa, ya que como reflejan las cardinalidades no todos los hombres ni todas las mujeres se encuentran casados.

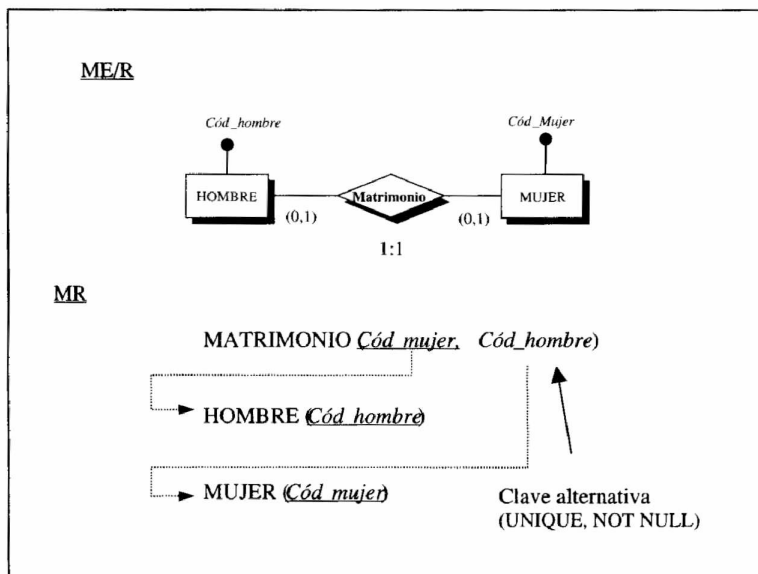


Figura 10.9. Transformación de una interrelación 1:1 en una relación

- b) Si una de las entidades que participa en la interrelación posee cardinalidades (0,1), mientras que en la otra son (1,1), conviene propagar la clave de la entidad con cardinalidades (1,1) a la tabla resultante de la entidad de cardinalidades (0,1). En la figura 10.10 tenemos una interrelación que recoge que un profesor que es responsable de un departamento, y supone, que un profesor puede ser responsable como máximo de un departamento y como mínimo de ninguno y que cada departamento tiene que tener siempre un responsable (pero sólo uno), en este caso propagamos la clave de PROFESOR a la tabla de DEPARTAMENTO, evitando así valores nulos y captando más semántica (recogemos la cardinalidad mínima 1, que en caso de realizar la propagación en sentido contrario no podríamos captar directamente).

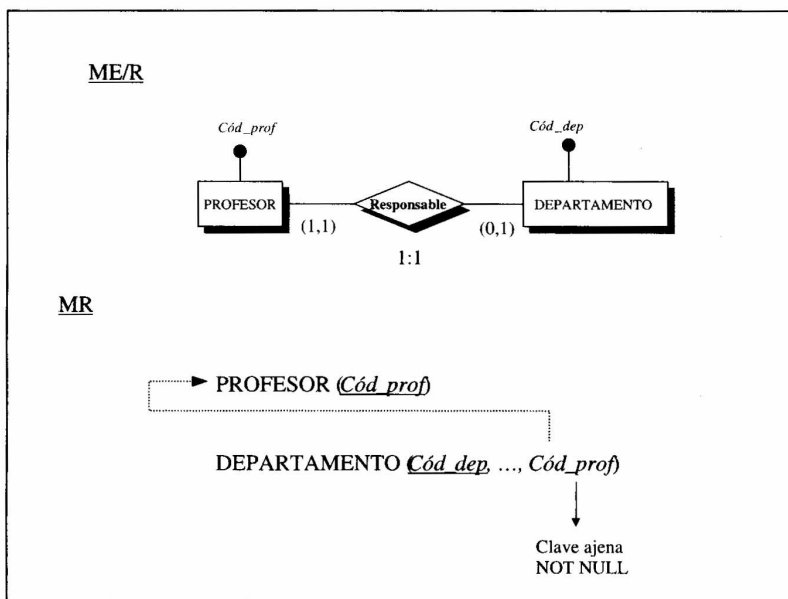


Figura 10.10. Transformación de una interrelación 1:1 por propagación de clave

- c) En el caso de que ambas entidades presenten cardinalidades (1,1), se puede propagar la clave de cualquiera de ellas a la tabla resultante de la otra, teniendo en cuenta en este caso los accesos más frecuentes y prioritarios a los datos de las tablas. Se puede plantear (también por motivos de eficiencia) la propagación de las dos claves, lo que introduce redundancias que deben ser controladas por medio de restricciones.

5. Transformación de atributos de interrelaciones. Si la interrelación se transforma en una relación, todos sus atributos pasan a ser columnas de la relación. Por ejemplo, la interrelación **Imparte** entre PROFESOR y CURSO de la figura 10.11 tiene un atributo *Núm_horas* (número de horas que imparte) pasa a ser una columna de la tabla que se crea a partir de ella. La transformación es directa y no hay pérdida de semántica.

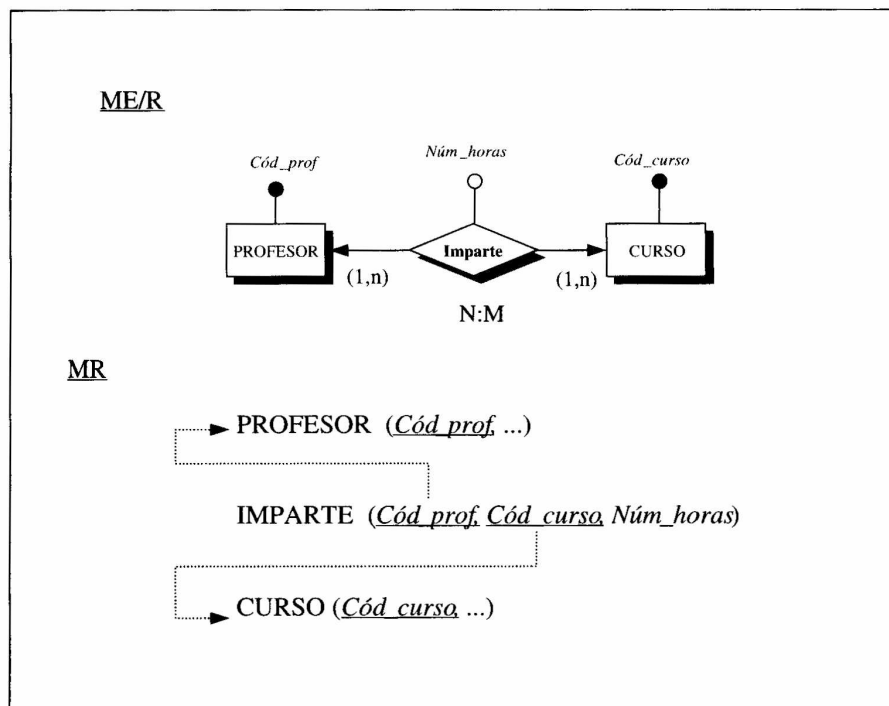


Figura 10.11. Transformación de los atributos de una interrelación en columnas de una tabla

En caso de que la interrelación se transforme mediante propagación de clave, sus atributos migran junto a la clave a la relación que corresponda, aunque ya hemos advertido que en este caso puede ser preferible crear una nueva relación para representar las interrelaciones que tienen atributos.

6. Transformación de restricciones. En cuanto a las restricciones de usuario, existen ciertas cláusulas en el LLS que pueden recogerlas. Por ejemplo, podemos restringir a un rango determinado los valores de un dominio a través de la cláusula **BETWEEN**, o bien determinar por enumeración los valores que puede tomar una columna en una tabla con la cláusula **IN**, como podemos observar en la regla 1 (véase Figura 10.3).

Otra posibilidad es utilizar la cláusula CHECK dentro de la descripción de la tabla para expresar una condición que deben cumplir un conjunto de atributos de la tabla, o la cláusula aserción si la comprobación afecta a atributos de más de una tabla. Por ejemplo, para que la fecha de inicio de un curso sea siempre menor que la de finalización, en la creación de la tabla tendríamos las siguientes sentencias SQL:

```
CREATE TABLE Curso (
  Cód_Curso Cursos,
  Nombre Nombres,
  Num_Horas Horas
  Fecha_I Fechas,
  Fecha_F Fechas,
  PRIMARY KEY (Cód_Curso),
  CHECK ( Fecha_I < Fecha_F));
```

Existe, además, la posibilidad de utilizar disparadores que, si bien no existen en el SQL92, sí se ofrecen en algunos productos.

4. REGLAS CONCERNIENTES A LAS EXTENSIONES DEL MODELO E/R

7. **Transformación de dependencias en identificación y en existencia.** Las dependencias en existencia y en identificación no son recogidas directamente en el MLS. En el ejemplo de la figura 10.12 vemos que la manera de transformar una interrelación de este tipo es utilizar el mecanismo de propagación de clave, creando una clave ajena, con nulos no permitidos, en la relación de la entidad dependiente, con la característica de obligar a una modificación y un borrado en cascada.

Además, en el caso de dependencia en identificación la clave primaria de la relación en la que se ha transformado la entidad débil debe estar formada por la concatenación de las claves de las dos entidades participantes en la interrelación. Así, el esquema E/R de la figura 10.12 da lugar al esquema relacional en SQL:

```
CREATE TABLE Curso
(Cód_Curso Códigos_cursos,
....,
PRIMARY KEY (Cód_Curso));
```

```
CREATE TABLE Edición
(Cód_Curso Códigos_Cursos,
```


Cód_Edición Códigos_Ediciones,
,
 PRIMARY KEY (Cód_Curso, Cód_Edicion)
 FOREIGN KEY (Cód_Curso) REFERENCES Curso
 ON DELETE CASCADE
 ON UPDATE CASCADE);

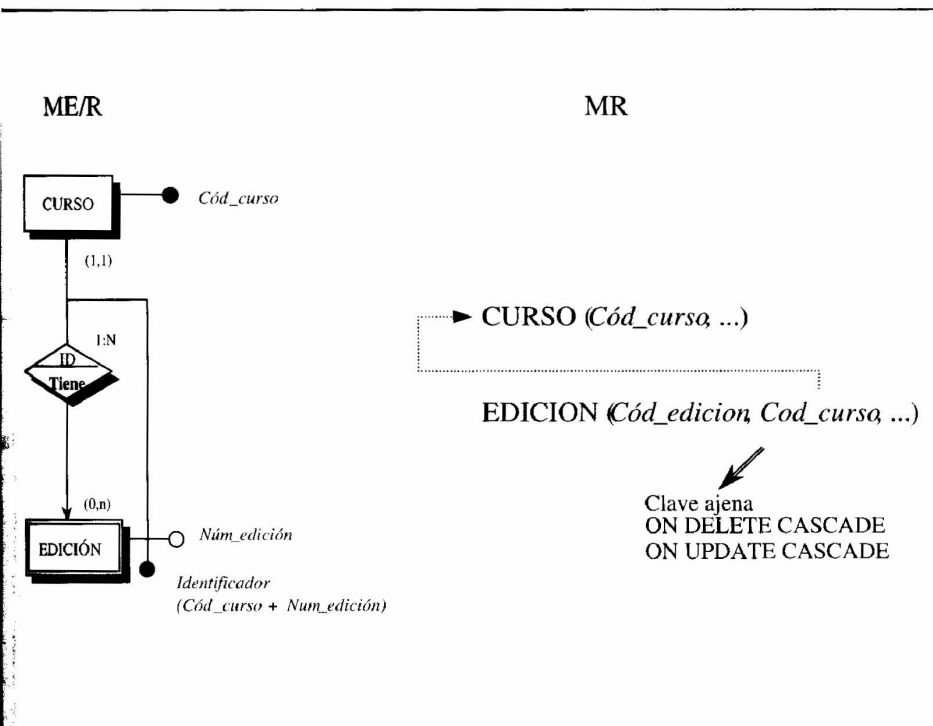


Figura 10.12. Transformación de una dependencia en identificación

- 8. Transformación de restricciones de interrelaciones.** Para soportar restricciones de interrelaciones (exclusión, inclusión, etc.) debemos definir las restricciones pertinentes en cada caso. Por ejemplo, en la figura 10.13 se muestra una restricción de exclusividad donde un profesor puede dirigir o impartir cursos, pero no ambos casos. Las interrelaciones **dirige** e **imparte** las resolvemos mediante el mecanismo de propagación de clave, llevando **cód_prof_dirige** y **cód_prof_imparte** a la relación **CURSO**.

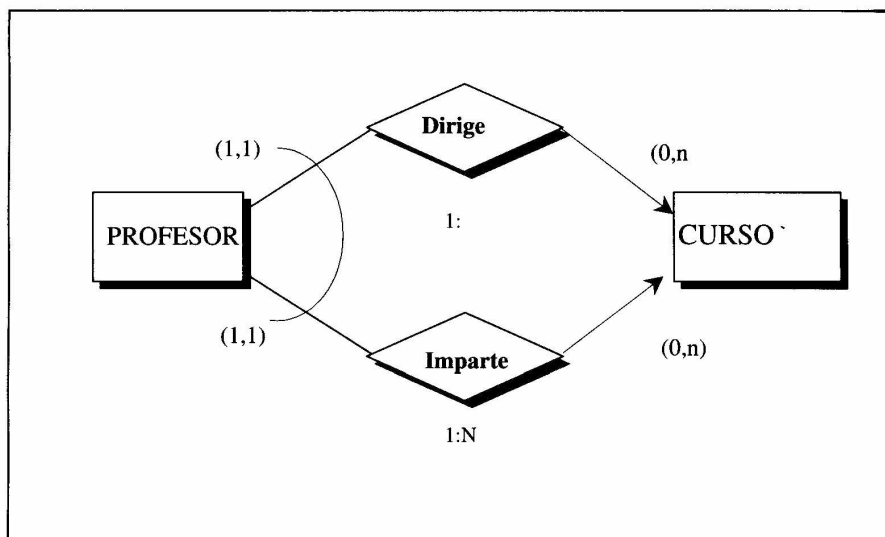


Figura 10.13. Restricción de exclusividad de interrelaciones

Para hacer que se cumpla la exclusividad habría que introducir las correspondientes restricciones en una cláusula CHECK, tal como se indica a continuación:

```

CREATE TABLE Curso (
  Cód_Curso Códigos_Cursos,
  Nombre Nombres,
  ....,
  Cód_prof_dirige Códigos_profesores,
  Cód_prof_imparte Códigos_profesores,
  PRIMARY KEY (Cód_Curso)
  FOREIGN KEY (Cód_prof_dirige) REFERENCES Profesor
    ON UPDATE CASCADE
  FOREIGN KEY (Cód_prof_imparte) REFERENCES Profesor
    ON UPDATE CASCADE
  CHECK (( Cód_prof_dirige NOT IN(SELECT Cód_prof_imparte
    FROM CURSO)
    AND Cód_prof_imparte NOT IN (SELECT Cód_prof_dirige FROM
    CURSO)));
  
```

Dejamos como ejercicio al lector resolver otros casos de interrelaciones exclusivas 1:N y de interrelaciones exclusivas N:M, así como otros tipos de

restricciones entre interrelaciones, mediante la definición de CHECK y/o de aserciones de forma análoga al ejemplo anterior.

9. Transformación de tipos y subtipos. En lo que respecta a los tipos y subtipos, no son objetos que se puedan representar explícitamente en el modelo relacional. Ante un tipo de entidad y sus subtipos caben varias soluciones de transformación al modelo relacional, con la consiguiente pérdida de semántica dependiendo de la estrategia elegida. Destacamos tres:

Opción a: Englobar todos los atributos de la entidad y sus subtipos en una sola relación. En general, adoptaremos esta solución cuando los subtipos se diferencien en muy pocos atributos y las interrelaciones que los asocian con el resto de las entidades del esquema sean las mismas para todos (o casi todos) los subtipos. Por ejemplo, la diferencia que existe entre un profesor que sea doctor y otro que no lo sea, podemos considerarla mínima teniendo en cuenta que ambos tienen los mismos atributos y ambos podrán impartir cursos (ver figura 10.14). Por este motivo, la solución adecuada en este caso sería la creación de una sola tabla que contenga todos los atributos del supertipo y los de los subtipos, añadiendo el atributo discriminante que indica el tipo de profesor.

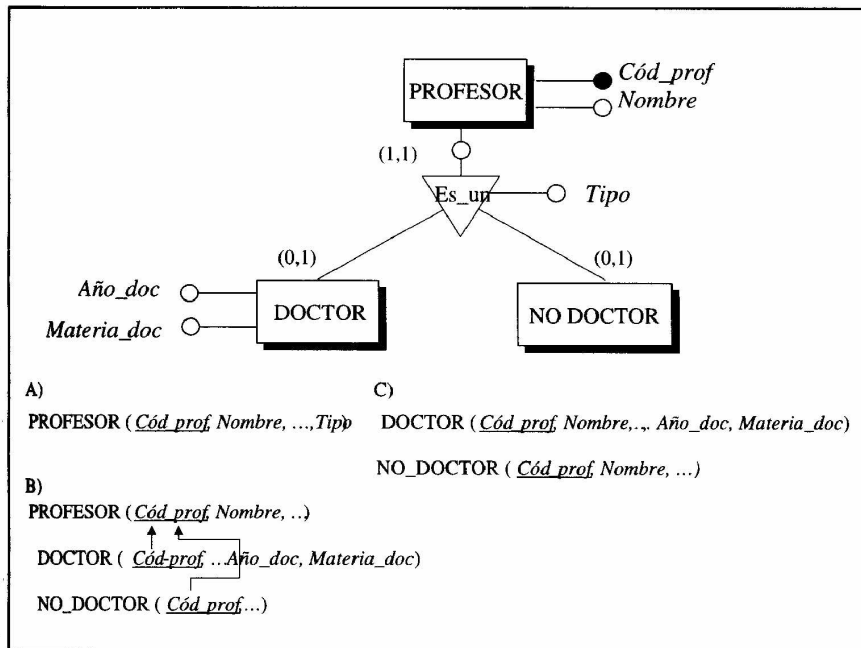


Figura 10.14. Transformación de subtipos

También habrá que especificar las restricciones semánticas correspondientes, por ejemplo:

```
CHECK ((Tipo = 'NO_DOCTOR'  
      AND Año_Doc IS NULL  
      AND Materia_Doc IS NULL))  
OR (Tipo= "DOCTOR"  
    AND Año_doc IS NOT NULL  
    AND Materia IS NOT NULL));
```

Hay que observar que el atributo discriminante de la jerarquía podrá admitir valores nulos en el caso de que la jerarquía sea parcial y que deberá declararse como NOT NULL si la jerarquía es total. Por otra parte, el atributo discriminante constituirá un grupo repetitivo, si los subtipos se solapan, debiendo, por tanto, separar este atributo en una relación aparte que tendrá como clave la concatenación de la clave del supertipo con el atributo discriminante; otra solución bastante más eficiente consiste en crear un código para los valores del atributo discriminante que contemple los posibles subtipos solapados.

Opción b: Crear una relación para el supertipo y tantas relaciones como subtipos haya, con sus atributos correspondientes. Ésta es la solución adecuada cuando existen muchos atributos distintos entre los subtipos y se quieren mantener de todas maneras los atributos comunes a todos ellos en una relación. Al igual que en el caso anterior, habrá que crear las restricciones y/o aserciones oportunas.

Opción c: Considerar relaciones distintas para cada subtipo, que contengan, además de los atributos propios, los atributos comunes. Se elegiría esta opción cuando se dieran las mismas condiciones que en el caso anterior —muchos atributos distintos— y los accesos realizados sobre los datos de los distintos subtipos siempre afectan a atributos comunes.

Podemos, por tanto, elegir entre tres estrategias distintas para la transformación de un tipo y sus subtipos al modelo relacional. Sin embargo, desde un punto de vista exclusivamente semántico, la opción b es la mejor. Por otra parte, desde el punto de vista de la eficiencia tenemos que tener en cuenta que:

Opción a: El acceso a una fila que refleje toda la información de una determinada entidad es mucho más rápido (no hace falta combinar varias relaciones).

Opción b: La menos eficiente, aunque, como ya hemos señalado, es la mejor desde un punto de vista exclusivamente semántico.

Opción c: Con esta solución aumentamos la eficiencia ante determinadas consultas (las que afecten a todos los atributos, tanto comunes como propios, de un subtipo) pero la podemos disminuir ante otras. Esta solución es en la que se pierde más semántica; además si existe solapamiento se introduce redundancia que debe ser controlada si queremos evitar inconsistencias. Hay que tener en cuenta que esta solución no es válida cuando la generalización es parcial.

Elegiremos una estrategia u otra dependiendo de que sea la semántica o la eficiencia la que prime para el usuario en un momento determinado.

Por lo que se refiere a la totalidad/parcialidad de la jerarquía y al solapamiento/disyunción de los subtipos, pueden ser soportados por medio de CHECK o aserciones, así como por disparadores o por procedimientos almacenados.

Por último, cabe destacar que en la última versión del SQL, la denominada SQL3:1999, se están definiendo más elementos a fin de soportar directamente la herencia por medio de las denominadas subtablas.

- 10. Transformación de la dimensión temporal.** En el caso de que en el esquema E/R aparezca el tiempo como un tipo de entidad, la transformación en el esquema relacional estándar no constituye mayor problema, ya que se tratará como otro tipo de entidad cualquiera y, por tanto se creará una relación más:

TIEMPO (*Fecha_I, Fecha_F, Hora_I, Hora_F, Minutos_I, ...*)

Sin embargo, cuando la dimensión temporal la hemos recogido en el esquema E/R a través de atributos de interrelación de tipo FECHA, la transformación en el MLS consiste en pasarlos a columnas de la relación que corresponda. Sobre este punto debemos tener cuidado a la hora de elegir la clave primaria de la relación resultante, dependiendo de los supuestos semánticos del entorno.

Por ejemplo, en la figura 10.15 podemos comprobar que la clave primaria de la relación obtenida de la interrelación "un socio toma prestado un libro de la biblioteca durante un periodo de tiempo determinado" no es sólo la concatenación del AIP de SOCIO Y el de LIBRO, sino que si se supone que un socio puede tomar prestado un mismo libro en distintos periodos de tiempo (por tanto *F_inicio* y *F_fin* son atributos multivaluados), es necesario, a fin de formar la clave primaria, añadir a los códigos de SOCIO y de LIBRO el atributo *F_inicio*. La transformación, por lo tanto, es directa, pero debemos tener en cuenta lo que acabamos de señalar respecto a la clave primaria a fin de salvaguardar la integridad de la base de datos.

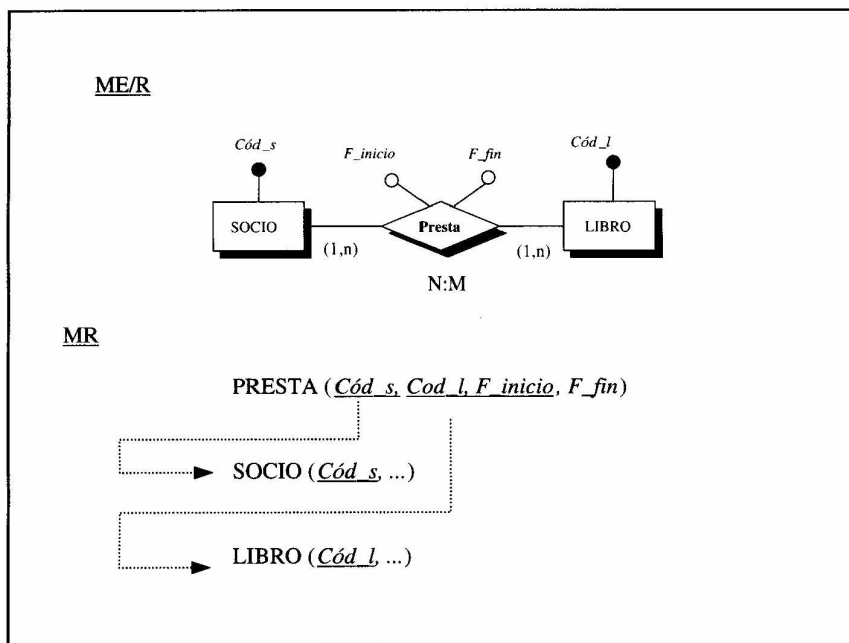


Figura 10.15. Transformación de la dimensión temporal

Es preciso observar que esta aparente sencillez en la transformación de algo tan complicado como es la dimensión temporal es debida a la poca semántica y precisión del modelo E/R (al igual que en el modelo relacional) para tratar los aspectos temporales. En sistemas de información en los que la dimensión temporal es fundamental, como es el caso de los sistemas estadísticos, el modelo E/R no resuelve muchos de los problemas de modelado relativos a la dimensión temporal.

11. **Transformación de Atributos Derivados.** No existe para los atributos derivados una representación directa y concreta en el MLE, sino que se pueden tratar como atributos normales, que pasarán a ser columnas de la relación que corresponda (véase figura 10.16).

En este caso es preciso construir un disparador que calcule el valor del atributo derivado cada vez que se inserten o borren las ocurrencias de los atributos que intervienen en el cálculo de éste y añadir la restricciones correspondientes. Por ejemplo, en el caso de la figura 10.16, cada vez que se inserte o borre una tupla en la tabla EDICIÓN, el disparador debe actualizar el atributo *N_ediciones* de la tabla CURSO.

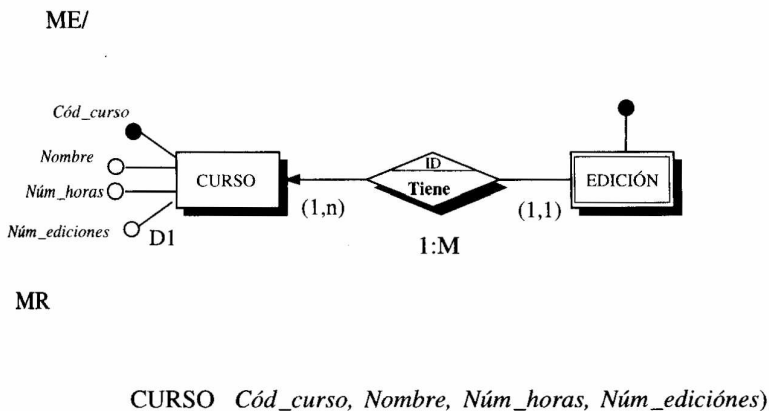


Figura 10.16.. Transformación de atributo derivado

Otra solución es no almacenar las columnas que provengan de atributos derivados, creando procedimientos que calculen los valores de éstos cada vez que se recuperan, lo que en ocasiones puede plantear problemas de semántica y en ocasiones de eficiencia.

5. GRAFO RELACIONAL

Una forma de representar gráficamente el esquema relacional de una manera sencilla y completa es el denominado *grafo relacional*, *diagrama esquemático*, SMITH (1985) o *grafo de combinación* SCHKOLNICK Y SORENSEN (1980).

Es un grafo compuesto de un conjunto de nodos multiparticionados, donde cada nodo representa un esquema de relación, es decir, una tabla de la BD. Para cada tabla, como mínimo, ha de aparecer su nombre y sus atributos, indicando su clave primaria subrayando los atributos que la componen con trazo continuo y sus claves ajenas (subrayando los correspondientes atributos por trazo discontinuo), véase figura 10.17.

Se dibuja, además, un conjunto de arcos que conectan los atributos que constituyen la clave ajena con la tabla referenciada, permitiendo así que el usuario encuentre los campos clave que comparten dominios comunes; en definitiva, los arcos representan la *referenciabilidad* de los atributos (clave ajena) de una relación respecto

a la clave primaria de la otra. Nosotros proponemos que los arcos estén direccionados de modo que el arco parta de la clave ajena y la flecha señale a la tabla referenciada.

GRAFO RELACIONAL

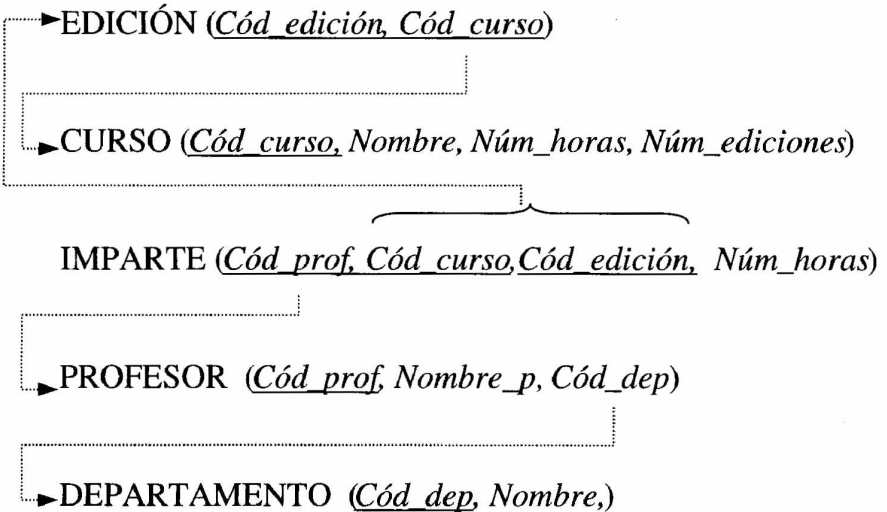


Figura 10.17. Ejemplo de grafo relacional

DISEÑO LÓGICO ESPECÍFICO Y DISEÑO FÍSICO

En este capítulo examinaremos las últimas etapas del diseño de bases de datos: el diseño lógico específico y el diseño físico. Ambas dependen del SGBDR que se utilice y, por tanto, nos limitaremos a ofrecer una visión global de las mismas tratando aspectos que son, en general, comunes a muchos productos. Se presenta, además, una introducción a la forma en que se organizan los datos en el almacenamiento secundario (nivel interno de ANSI, incluso, nivel físico), lo que constituye el fundamento del diseño físico.

1. DISEÑO LÓGICO ESPECÍFICO

A partir del esquema lógico estándar (ELS) obtenido en la etapa anterior del diseño lógico, y teniendo en cuenta el modelo lógico específico (MLE) en el que se va a instrumentar la base de datos, se elabora el esquema lógico específico (ELE), que será descrito en el lenguaje de definición de datos del producto comercial que estemos utilizando.

La transformación del ELS al ELE lleva consigo un conocimiento del SGBD que va a ser aplicado a fin de:

- Ver en qué grado soporta el modelo relacional y, por tanto, el modelo lógico estándar (MLS)
- Adaptar el ELS a las características propias del producto que se va a utilizar (a su MLE).
- Definir el ELE en la sintaxis propia del SGBD.

Se debe estudiar la correspondencia entre los conceptos del MLS y los del SGBD concreto, pudiéndose dar los siguientes casos:

- **El SGBD soporta todos los conceptos del MLS sin restricciones.** La transformación del ELS al ELE es prácticamente directa, sólo se ha de describir (simplemente transcribir) el esquema lógico en la sintaxis propia del SGBD utilizado, que será normalmente bastante parecida a la del lenguaje SQL.
- **El SGBD no soporta ciertos conceptos, o bien los soporta pero con restricciones.** Tendremos que utilizar entonces nuevos objetos (como índices), realizar una programación complementaria, incluyéndola en la metabase; o bien, en último caso, transferir a los programas restricciones no soportadas convenientemente por el MLE. Esto ocurre, por ejemplo, en algunos SGBDR con respecto a la integridad referencial.

El modelo de referencia de ANSI (1986) nos advierte sobre los inconvenientes que puede suponer el trasladar a los programas la semántica de los datos y aconseja que se procure almacenar de forma centralizada el control de integridad de la base.

2. IMPLEMENTACIÓN DE LOS PRINCIPALES CONCEPTOS DEL MODELADO RELACIONAL

En este epígrafe vamos a ver la transformación del ELS que se ha obtenido por la aplicación de las reglas del capítulo anterior, al ELE. Únicamente vamos a analizar aquellos aspectos en los que es necesaria una transformación adicional debido a que los conceptos del MLS no son soportados por el MLE del producto.

2.1. Dominios

Casi ningún producto ofrece ni siquiera la sintaxis para la definición de dominios, pero aun cuando dispongan de la sintaxis correcta, no realizan con precisión las funciones semánticas completas que un dominio tiene encomendadas, como son, según CODD (1990):

1. Comprobar que la base de datos es consistente y se mantiene integrada, ya que los valores de los datos se extraen de *fuentes* comunes.
2. Declarar una sola vez cada tipo de datos permitido en el esquema.

3. Soportar la integridad y consistencia de los dominios entre sí. Por ejemplo, a la hora de realizar operaciones compatibles en el dominio, como combinación, unión, intersección, etc.
4. Posibilitar la creación de operadores y de características propias de los dominios.
5. Simplificar transacciones complejas sobre varias columnas que pertenecen al mismo dominio, haciendo la transacción sobre el dominio directamente.
6. Facilitar la definición de comprobaciones en el SGBD.
7. Hacer posible la indización directamente sobre los dominios, no sobre las columnas de las tablas.

Al no disponer, la mayoría de los SGBDR comerciales, de sentencias para la definición de dominios, es al definir la columna de una tabla cuando se especifica el tipo de dato, la longitud y (si el esquema lo permite) las restricciones pertinentes. Así, por ejemplo, no podríamos definir un dominio como el de los DNI, por lo que en el ELE no podría aparecer el dominio de los DNI que se encontraba en el ELS. Entonces, en el momento de la creación de la tabla que tenga un atributo definido sobre el dominio de los DNI debería indicarse que se trata de enteros con longitud máxima de 9 dígitos, con toda la pérdida de semántica que esto supone.

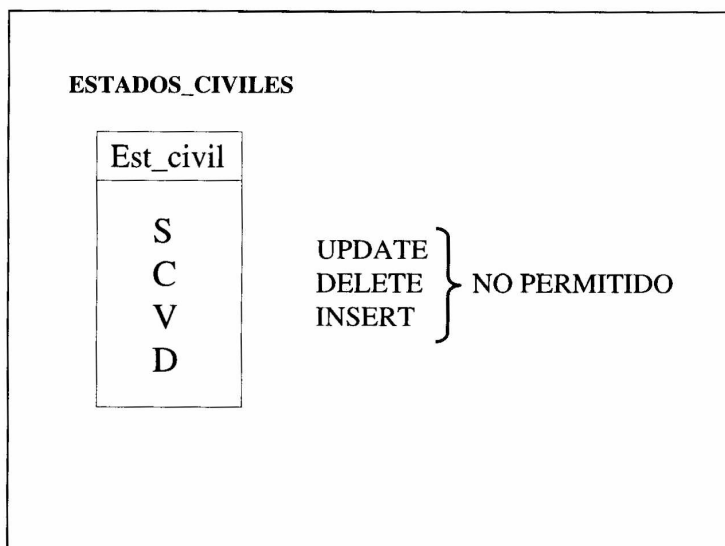


Figura 11.1. Representación de una relación de dominio

Una simulación, si bien muy pobre, de los dominios es la construcción de un procedimiento que compruebe que los valores que se pretenden insertar o modificar se encuentran en una relación de una sola columna que no sea susceptible de inserción, borrado o modificación; es decir una tabla estática que sólo el administrador podrá modificar si lo cree necesario. Podríamos también utilizar, en lugar de una relación de una sola columna, las estructuras de datos que ofrece el lenguaje en el que se genera el procedimiento (por ejemplo, vectores, matrices o simples variables). En la figura 11.1 se muestra la tabla que define los valores Estados_civiles.

En cuanto a la utilización de estas relaciones, denominadas por algunos autores *tablas de dominio*, hay que resaltar que, como señala DURELL (1991), las organizaciones deberían promover su utilización, ya que:

- Permiten aislar los datos de los procesos.
- Facilitan que los datos puedan ser compartidos.
- Permiten estandarizar la edición y validación de datos.
- Promueven la implicación de los usuarios como responsables de los datos.
- Se mejora la extensibilidad de los datos, al ser fácil añadir información a las tablas de dominios.

Pero la simulación de los dominios a través de tablas no es suficiente, ya que somos conscientes de que la verdadera función del dominio, formalmente hablando, no es sólo la de *prestar* sus valores a las columnas de las relaciones de una base de datos. Otras funciones, como la de controlar que determinadas operaciones no se pueden efectuar sobre la base de datos sin atentar contra la integridad de la misma, deberían ser simuladas igualmente.

Otro inconveniente de esta solución es que transfiere a los procedimientos una semántica que debería estar incluida en los datos. Algunos SGBD resuelven este problema al permitir definir reglas y *disparadores* directamente asociados a los datos (en la metabase), de modo que los procedimientos no puedan *saltárselos*.

2.2. Claves primarias

Casi todos los productos actuales recogen este concepto con sintaxis muy parecida a la del SQL estándar, pero todavía algunos de ellos se quedan sólo en la sintaxis, obligando a crear un índice único sobre los atributos que componen la clave y a declarar que los nulos no están permitidos.

Cuando el producto no dispone de la sintaxis para la definición de las claves primarias, el procedimiento a seguir es el siguiente:

1. Especificar cada campo de la clave primaria para que no admita valores nulos (esto es, como NOT NULL).
2. Definir, cuando se crea la tabla, un índice de tipo único (UNIQUE) sobre la combinación de todos los campos de la clave primaria.
3. Asegurar la existencia de este índice, creándolo con la tabla y no borrándolo en tanto no se borra la tabla.
4. Mantener las especificaciones de CLAVE PRIMARIA como comentario en el catálogo del SGBDR en el caso de que ni siquiera se permita la definición sintáctica.

Incluso los productos que recogen el concepto de clave primaria, tanto en su sintaxis como en su semántica, no obligan a su definición (al igual que el SQL estándar). La razón de ello es la compatibilidad con versiones anteriores que no disponían de la correspondiente sentencia.

2.3. Claves ajenas

El modelo relacional específico de algunos sistemas comerciales, hoy por hoy, no recoge en su totalidad el concepto de clave ajena, siendo un problema similar a lo que ocurre respecto a la clave primaria, ya que la sintaxis de algunos productos la admite, pero sin efecto semántico alguno al no comprobarse su violación, pudiéndose atentar, por tanto, contra la integridad de los datos. Varios sistemas sólo recogen la semántica de ciertas opciones de clave ajena, por ejemplo, únicamente admiten la modificación restringida. En otros casos, aunque el producto no incorpora la sintaxis de clave ajena en la sentencia de definición de tablas, sí proporciona la posibilidad de definir procedimientos almacenados o disparadores que permiten implementar este concepto, aunque no se puede programar de manera general, y habrá que escribir el procedimiento para cada clave ajena que se necesite declarar.

Para los sistemas que no ofrecen la semántica de la clave ajena, los pasos a seguir son los siguientes:

1. Introducir las restricciones de clave ajena como requisito de especificación de los programas.
2. Si los nulos no están permitidos en la clave ajena, especificar los campos que la componen como NOT NULL.
3. Mantener las especificaciones de CLAVE AJENA como comentario en el catálogo, o bien incluir su definición sintáctica si está permitida.

4. Utilizar los mecanismos de seguridad del SGBDR para prohibir las operaciones de actualización efectuadas de forma interactiva por el usuario final que puedan violar las restricciones de clave ajena, como son:
 - Borrado de filas que tengan una clave referenciada
 - Modificación de una clave primaria referenciada
 - Inserción en la tabla que referencia
 - Modificación de la clave ajena en la tabla que referencia.
5. Para conservar la integridad ante cualquier evento (pensando en que alguno de los procedimientos anteriores pudiera fallar debido a un error de los programas o a un fallo del sistema) es aconsejable escribir un programa que, ejecutado periódicamente, compruebe y notifique posibles violaciones de integridad.
6. Por motivos de eficiencia, y dependiendo del SGBDR que se utilice, considerar la creación de un índice sobre la combinación de campos de la clave ajena, o la utilización de otro tipo de estructuras físicas, como agrupamientos (*cluster*).

En cuanto a la implementación de la clave ajena y la integridad referencial, hay que tener en cuenta que en lo visto hasta ahora hemos dado prioridad ante todo a la *semántica*, pero no podemos olvidar la *eficiencia*, y, desafortunadamente, el mecanismo para la comprobación de la integridad referencial penaliza los tiempos de respuesta de las aplicaciones.

De hecho, basta con pensar que el borrado de un registro de una tabla puede provocar el borrado (en cascada) de varios registros de otras tablas, y así sucesivamente, con el consiguiente aumento de tiempo, lo que es grave en caso de uso interactivo del sistema. Puede ser, por tanto, inviable en ciertas aplicaciones y bajo determinadas circunstancias (como grandes volúmenes de datos, *hardware* poco eficiente, etc.) implementar la integridad referencial en *línea*, siendo preferible hacerlo en un programa que se ejecute en diferido (por lotes) y no de forma interactiva.

En este caso se van almacenando los cambios que afectan a la integridad referencial en la base de datos, aprovechando cuando el sistema esté menos cargado para ejecutar un procedimiento que vaya comprobando y realizando las acciones pertinentes. Como contrapartida a la mejora de la eficiencia tenemos que ser conscientes de que la base de datos queda semánticamente inconsistente durante ciertos periodos de tiempo.

2.4. Otros conceptos del modelo relacional

Para simular otros conceptos del modelo relacional no incluidos en los productos, normalmente hay que introducir disparadores o procedimientos que realicen la comprobación de las restricciones de integridad que se han definido en la etapa de diseño lógico estándar.

Como ya hemos señalado, hay sistemas que permiten almacenar estos procedimientos en el propio diccionario, mientras que otros nos obligan a llevar estas restricciones a los procesos.

3. OBJETIVOS Y ACTIVIDADES DEL DISEÑO FÍSICO

La última etapa del proceso es la del diseño físico, en el cual, teniendo presentes requisitos de los procesos, características de los SGBD, del SO y del hardware, se pretenden entre otros, los siguientes objetivos:

- Disminuir los tiempos de respuesta.
- Minimizar el espacio de almacenamiento.
- Evitar las reorganizaciones.
- Proporcionar la máxima seguridad.
- Optimizar el consumo de recursos.

En definitiva, cumplir los objetivos del sistema y conseguir la optimización del *ratio* coste/beneficio.

Las entradas y salidas del diseño físico se representan en la figura 11.2. En ella, como entradas, además de los objetivos de diseño físico (con sus correspondientes prioridades y cuantificados en lo posible), aparece el resultado de la etapa de diseño lógico (es decir, el esquema lógico), así como los recursos de máquina y de software (como el sistema operativo) disponibles, además de información sobre las aplicaciones que es preciso tener en cuenta a fin de optimizar aspectos como el tiempo de respuesta¹ y definir ciertas políticas como la de seguridad.

A partir de estas entradas, en la etapa de diseño físico, se producirán, como salidas, una estructura interna (o vista del sistema) junto con especificaciones que

¹ El lector interesado puede consultar TEOREY y FRY (1982) para conocer cómo se calculan los tiempos de respuesta de distintas estructuras y métodos físicos, información que deberá completar con el manual de administración del producto en el que realice la implementación.

sirvan para realizar el ajuste o *tunning* de la base, así como las normas relativas a la seguridad de la misma.

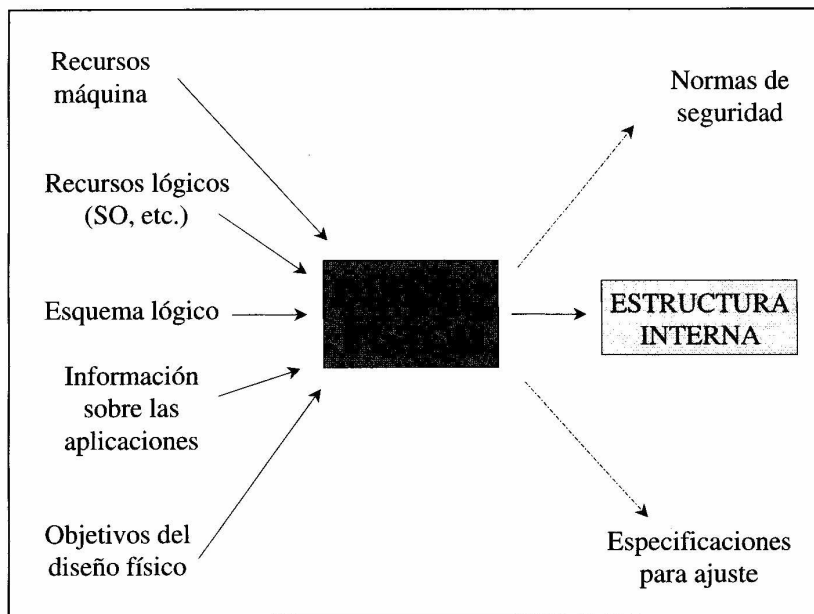


Figura 11.2. Entradas y salidas del proceso del diseño físico

En general, existen tres *estrategias* de los fabricantes en cuanto al diseño físico:

- El SGBD **impone** una estructura interna, dejándole al diseñador muy poca flexibilidad, lo que suele aumentar la independencia físico/lógica, pero disminuye la eficiencia.
- El administrador **diseña** la estructura interna, lo que en general supone una importante carga para el administrador y puede influir negativamente en la independencia, aunque puede mejorar la eficiencia.
- El SGBD **proporciona** una estructura interna a partir de algunos parámetros que le proporciona el diseñador, el cual, posteriormente, puede irlos modificando a fin de realizar su ajuste (*tuning*) y optimizar, así, el rendimiento de la base de datos.

Esta última *estrategia* tiene una serie de ventajas:

- La BD puede empezar a funcionar de inmediato.

- La eficiencia va aumentando al irse efectuando sucesivos ajustes.
- La independencia físico/lógica se mantiene.

Este último enfoque, que es el que se suele ofrecer en la mayoría de los SGBD actuales, es el que mejor se adapta a la metodología propuesta.

Como señala CERI (1983), *“El problema del diseño físico para el administrador de la BD consiste en proveer un conjunto eficiente de estructuras de acceso, de modo que el optimizador pueda tomar las mejores decisiones”*. El administrador (o el diseñador responsable del diseño físico) ha de elegir entre todas las opciones disponibles, aquellas técnicas de estructuración física que permitan un acceso más eficiente, dados los requisitos concretos del correspondiente sistema de información.

Distintos estudios muestran que del orden del 80% al 90% de las manipulaciones sobre la BD son realizadas únicamente por un 10% a 20% de las aplicaciones, por lo que el estudio de las frecuencias de éstas puede ayudarnos a la hora de elegir las mejores estrategias respecto a los caminos de acceso.

Una vez diseñadas las aplicaciones se conocerá cuáles son las consultas más frecuentes y/o prioritarias a la base de datos, por lo que será conveniente crear las estructuras físicas que ayuden a localizar las filas seleccionadas en dichas consultas y a reducir así los accesos a disco.

Entre los instrumentos más importantes (y generales) del diseño físico se encuentra la selección de los índices secundarios, que es uno de los problemas clave en la implementación física de una base de datos.

Dependiendo del producto que estemos utilizando podremos indicar ciertas características adicionales del índice, como son si se encuentra o no comprimido, el orden, etc.

Existen otros elementos importantes en el diseño físico, aunque no todos los sistemas comerciales disponen de ellos, o si disponen de los mismos, a veces no se le da al diseñador la oportunidad de actuar sobre los mismos ajustándolos a cada caso concreto. Algunos de estos elementos son:

- Registros físicos
- Punteros
- Direccionamiento calculado (*hashing*)
- Agrupamientos (*cluster*)
- Bloqueo y compresión de datos
- Asignación de espacios de almacenamiento como memorias intermedias (*buffers*)

- Asignación de conjuntos de datos a particiones y a dispositivos físicos
- Etc.

Sin embargo, la escasa flexibilidad en cuanto a las estructuras físicas que, en general, ofrecen los sistemas comerciales, obliga en determinados casos a llevar a cabo un proceso de reestructuración de relaciones (desnormalización, particionamiento vertical, etc.) para conseguir tiempos de respuesta aceptables. Por tanto, se deberá proceder de forma iterativa desde el diseño lógico específico al diseño físico, y viceversa, hasta optimizar el anterior ratio.

Desafortunadamente, y a pesar de que se han llevado a cabo distintos trabajos de investigación, no existe un modelo formal para el diseño físico (análogo, por ejemplo, al modelo relacional para el diseño lógico), por lo que éste resulta hasta ahora, muy dependiente del producto comercial concreto.

Es posible, sin embargo, hacer unas consideraciones de carácter general (independientes de productos comerciales) respecto a los conceptos generales de organización de archivos en los que se ha de apoyar el diseño físico, y que se expondrán a continuación.

4. CONCEPTOS GENERALES DE ALMACENAMIENTO DE LOS DATOS EN SOPORTE SECUNDARIO

Los datos de la base de datos (que pueden llegar a volúmenes de giga o terabytes) se han de almacenar físicamente de forma permanente en soporte no volátil². Es necesario una constante transferencia de los datos ente el soporte permanente y la memoria principal; esta transferencia es lenta comparada con la velocidad de la unidad central, por lo que es necesario que la organización de los datos en soporte secundario se haga de forma que se minimice dicha transferencia a fin de que el sistema proporcione tiempos de respuesta aceptables.

Por todo ello, las organizaciones físicas de las bases de datos se han ido complicando en el transcurso del tiempo a fin de mejorar la eficacia de los sistemas; el administrador de la base de datos ha de conocer bien las características del producto que está utilizando a fin de elegir aquellas opciones que optimicen recursos y tiempos de respuesta.

² Existe un jerarquía con varios niveles de almacenamiento, que se distinguen principalmente por su volatilidad o permanencia y por su velocidad de transferencia; van desde la memoria "caché" y la memoria principal como dispositivos de almacenamiento primario (es decir, directamente accesibles desde la unidad central de proceso) hasta los discos magnéticos, discos ópticos y cintas magnéticas, entre otros, como dispositivos de almacenamiento secundario (no accesibles directamente). A veces los discos ópticos y la cinta magnética reciben el nombre de almacenamiento terciario.

4.1. Diseño de bloques y gestión de almacenamiento intermedio

Los datos de una base de datos, en su nivel intermedio, se organizan en archivos (colecciones de registros) que se almacenan en disco magnético³.

La unidad básica del archivo es el registro físico, también denominado registro almacenado, página o bloque, que es la unidad de datos mínima que puede tratarse en una operación de entrada/salida.

Un bloque puede contener varios registros lógicos (filas en el caso del modelo relacional); denominándose factor de bloqueo de un archivo al número de registros lógicos por bloque para dicho archivo. También puede ocurrir, como en el caso de registros lógicos muy grandes, que cada uno de ellos se encuentre repartido en varios bloques.

El tamaño de los bloques depende del producto específico de que se trate, así como del sistema operativo subyacente, siendo habitual encontrar bloques de tamaño de entre 2 y 4 Kbytes.

Los bloques se encuentran almacenados en sectores de discos, y deben ser accedidos por el sistema de gestión de la base de datos empleado para ello, en general y en mayor o menor medida, los mecanismos de gestión de archivos del sistema operativo⁴. El problema es que el tiempo para acceder a disco es bastante elevado, estando compuesto por:

- Tiempo de búsqueda (*seek*), necesario para desplazar las cabezas al cilindro⁵ requerido (es el tiempo más elevado, ya que obliga a un desplazamiento mecánico de la cabeza de lectura/escritura hasta situarse en el cilindro que corresponda).
- Tiempo de latencia o retardo rotacional, que emplea el sector en su giro hasta posicionarse debajo de la cabeza.
- Tiempo de transferencia, necesario para transportar los datos de un sector del disco a memoria principal (es mucho menor que los dos anteriores).

Para acelerar la recuperación de datos de la base, se emplea una memoria de almacenamiento intermedio (*buffer*), donde se transfieren con antelación los datos que van a ser utilizados, evitando así accesos a disco. La gestión de esta memoria, en los

³ Aunque, como hemos indicado, existen distintos dispositivos de almacenamiento no volátil, nos vamos a referir únicamente al disco magnético que es el más habitual en el caso de las bases de datos.

⁴ En algunos casos el SGBD asume prácticamente todas las funciones de acceso a los datos, mientras que en otros se apoya mucho más en el sistema operativo utilizando los métodos de acceso del mismo.

⁵ Se entiende por cilindro el conjunto de pistas del mismo diámetro.

sistemas operativos, suele realizarse mediante políticas como la conocida LRU (*Least Recently Used*) que, como su nombre indica, establece que si es necesario eliminar un bloque de la memoria intermedia con el fin de hacer espacio para otro, se escoge el bloque utilizado menos recientemente; en los SGBD se lleva a cabo mediante otro tipo de estrategias como la MRU (*Most Recently Used*) asociada a la técnica de registro *clavado*. La aplicación de políticas de sustitución distintas en las bases de datos que en los sistemas operativos es posible porque en las primeras se tiene más información sobre qué datos van a ser próximamente procesados.

En algunos SGBD el administrador puede especificar las características de los bloques; así, los bloques se suelen agrupar de forma contigua en unidades mayores (extensiones), que pueden ubicarse en diferentes partes del disco, permitiendo también separar el almacenamiento de los datos de una tabla de las estructuras de acceso (por ejemplo, índices).

Dentro de los parámetros que se pueden especificar en algunos productos destacamos:

- El porcentaje de espacio libre de cada bloque que se reserva para futuras modificaciones de las filas de la tabla (PCTFREE). Hay que tener en cuenta que, cuando se actualizan datos de un bloque, puede ocurrir que los nuevos valores no “entren” en el espacio que ocupaban los antiguos, con lo que el sistema puede verse obligado a concatenar bloques, disminuyendo así los tiempos de respuesta y desaprovechándose espacio disponible. Si se deja más espacio libre, se evitan reorganizaciones de la base de datos.
- Porcentaje de utilización de cada bloque (PCTUSED). Si este valor es bajo, se aumenta el espacio no utilizado de la base de datos, pero se reduce el procesamiento necesario para el borrado de datos, mientras que aumenta para la inserción.
- Número de bloques que se asignan a las extensiones de una tabla, bien de forma inicial (INITIAL), bien en las sucesivas ampliaciones (NEXT), pudiéndose también definir el porcentaje de crecimiento de las extensiones (PCTINCREASE) y un número máximo de éstas (MAXEXTENTS).

Un ejemplo de definición de estos parámetros (en el SGBD ORACLE) para una tabla (Autores) podría ser el siguiente:

```
CREATE TABLE Autores (  
  Código      NUMBER(4)      PRIMARY KEY,  
  Nombre      VARCHAR2(25) NOT NULL,  
  Institución  VARCHAR2(20),
```

```
PCTFREE 20
PCTUSED 15
TABLESPACE DATOS1
STORAGE
  (INITIAL 100K
   NEXT 200K
   MAXEXTENTS 10
   PCTINCREASE 30));
```

5. ORGANIZACIÓN DE ARCHIVOS Y MÉTODOS DE ACCESO

La organización de un archivo determina el modo de estructurar los registros pertenecientes a un archivo en almacenamiento secundario. Los métodos de acceso nos permitirán localizar dichos registros.

Estos dos conceptos, aun siendo distintos, están estrechamente relacionados. La elección de una determinada organización permitirá realizar distintos tipos de acceso a los registros del archivo y, viceversa, si se desea acceder a un registro o conjunto de registros de una determinada forma, se deberá elegir una organización adecuada para ello.

La organización de los archivos que constituyen la base de datos y los métodos de acceso repercuten fuertemente en los tiempos de respuesta de los SGBD.

Los métodos básicos de acceso son los siguientes:

- **Acceso secuencial.** Para localizar un registro se necesita haber accedido al registro anterior. Puede resultar el modo de acceso más eficiente cuando se tiene que procesar el archivo completo (o un porcentaje del mismo).
- **Acceso directo.** Se localiza un registro por su dirección, obtenida a partir del valor de una clave de direccionamiento o por la posición relativa que ocupa el registro en el archivo.

La elección de un tipo de organización se realizará para cada tabla de la base de datos, y dependerá de tres factores:

1. Del tipo de proceso: Más concretamente del tipo de acceso requerido (secuencial o directo) y de si las operaciones que se realizan en el archivo son de recuperación o de actualización.
2. De los factores que se desean optimizar: Estos factores son diversos y están muy relacionados entre sí, pero podemos enumerar los siguientes: espacio de almacenamiento, tiempos de respuesta, accesos a disco, etc.

3. De las características del archivo: Es decir, si el archivo tiene gran volumen o no, si su crecimiento será rápido, si se eliminarán muchos registros, etc. En definitiva, tamaño, volatilidad y crecimiento, principalmente.

A continuación haremos una breve introducción⁶ a los distintos tipos primarios de organizaciones, presentando las características de cada una de ellas, las ventajas e inconvenientes de su utilización, los tipos de proceso para los que son adecuadas y los factores que optimiza.

5.1. Organizaciones consecutivas

Las organizaciones consecutivas sitúan los registros físicamente uno a continuación de otro. Si la colocación de los registros, no tiene un orden lógico, el tipo de organización se denomina **consecutiva serial**; si, por el contrario, tienen un orden, según una clave de ordenación, se denomina **consecutiva secuencial**.

En este tipo de organizaciones, el método de acceso que se utiliza para localizar los registros almacenados en el archivo no puede ser otro que el secuencial si la organización es serial (sin orden); si es secuencial (con orden) puede utilizarse, además, un acceso directo por posición relativa del registro (**búsqueda binaria**, también llamada **dicotómica**).

La organización consecutiva es adecuada, en general, para recuperar archivos en procesos por lotes (en los que hay que procesar un porcentaje elevado de los registros del archivo); optimizan al máximo el espacio de almacenamiento, son muy fáciles de programar, pueden utilizar registros de longitud variable y heterogéneos, es decir, de distintos tipos (en este caso la búsqueda no puede ser binaria) y no está limitada al tipo de soporte directo (pueden utilizarse también con cinta magnética, aunque no es habitual).

Es conveniente utilizar este tipo de organización cuando existe una carga masiva de datos, las tablas son pequeñas, o cuando, como ya hemos indicado, en el proceso del archivo se accede normalmente a casi todas las filas (en este caso un índice disminuye la eficiencia, por lo que, en algunos sistemas, se puede inutilizar un índice simplemente sumando un cero a los valores numéricos o concatenando un blanco a los caracteres, consiguiendo de esta forma influir en el optimizador del SGBD.

⁶ Un estudio más profundo de estos temas se puede encontrar, entre otras muchas obras, en TEOREY y FRY (1982), FOLLE (1992), WIEDERHOLD (1985), LOOMIS (1989), SMITH (1987), LIVADAS (1990), etc.

11.2. Organizaciones direccionadas

La necesidad de realizar un acceso rápido y directo a unos determinados registros obliga a buscar otras alternativas distintas a las organizaciones consecutivas, y una de ellas es la utilización de organizaciones direccionadas. Este tipo de organización almacena los registros según una relación establecida entre el valor de la clave de direccionamiento y la dirección física del registro, siendo posible realizar un acceso directo a un registro si se conoce la clave de direccionamiento, aunque puede impedir (o al menos dificultar) el acceso secuencial.

La ubicación de un registro se determinará aplicando un **algoritmo de transformación** a la clave de direccionamiento, lo que dará como resultado la *dirección base* donde se debería almacenar dicho registro, véase figura 11.3.

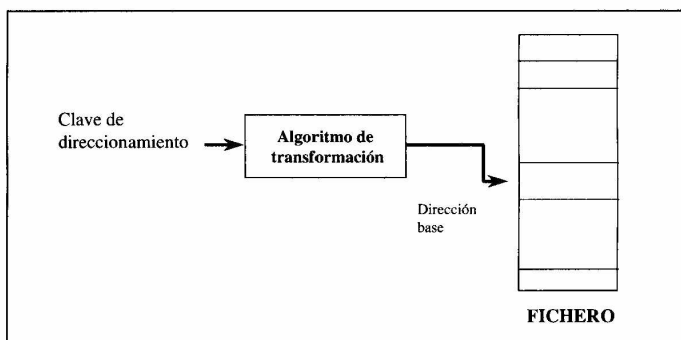


Figura 11.3. Obtención de la dirección base de un registro a partir de la clave de direccionamiento

La dirección base estará comprendida dentro de un rango de valores, que representan las direcciones relativas, denominado espacio de direccionamiento, el cual se fija previamente en función del volumen de datos que hay que almacenar. La organización direccionada resulta muy adecuada en procesos de tipo selectivo donde hay que acceder a un registro por el valor de la clave de direccionamiento o por un rango de la misma.

Dentro de este tipo de organización se puede distinguir entre organizaciones direccionadas **directas** y **dispersas**.

Direccionadas directas

La función de transformación en esta organización es 1:1, lo que implica que no puede dar direcciones base repetidas; además, la clave de direccionamiento y el algoritmo de transformación deben ser tales que las direcciones base obtenidas han de

estar comprendidas dentro de un espacio de direccionamiento fijado de antemano. La clave de direccionamiento no podrá tener, por tanto, valores repetidos.

Las ventajas de utilizar este tipo de organización es que los registros siempre están ordenados por la clave de direccionamiento, estando los registros situados en direcciones contiguas, por lo que el acceso puede ser secuencial ordenado, además de directo. Todo registro ocupa su dirección base, lo que permite que, mediante un solo acceso a memoria secundaria, localicemos un registro.

El inconveniente que tiene la utilización de este tipo de organización es que las exigencias de la clave de direccionamiento son muy restrictivas, por lo que suele ser muy difícil encontrar una clave con estas características.

Direccionadas dispersas

Las organizaciones direccionadas dispersas utilizan una función de transformación $n:1$, lo que significa que para distintas claves de direccionamiento podemos tener la misma dirección base, produciéndose entonces una colisión. Estos registros se denominan sinónimos, véase figura 11.4.

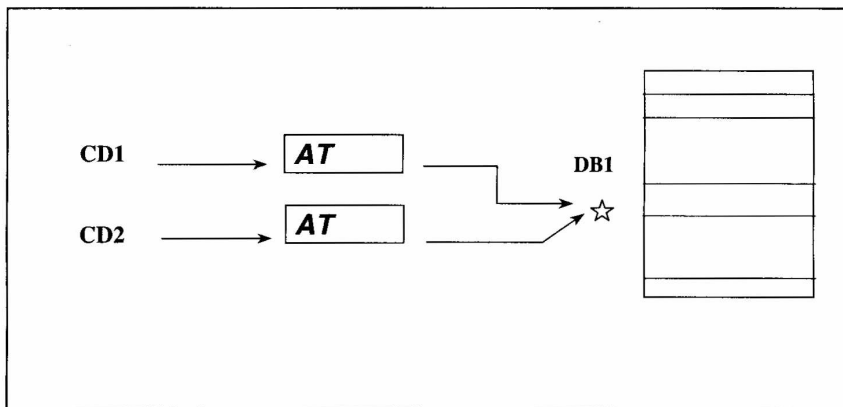


Figura 11.4. Problema de las colisiones: los registros con clave de direccionamiento CD1 y CD2 dan la misma dirección base

Por lo tanto, para este tipo de organizaciones debemos tomar las siguientes decisiones:

1. El tamaño del bloque o registro físico en el espacio de direccionamiento. Si el tamaño de bloque es uno el direccionamiento es a registro y si por el contrario es mayor que uno, el direccionamiento es a cubo.

2. Fijar un espacio de direcciones, considerando que no puede haber un densidad de registros mayor del 80%. Donde la densidad es:

Densidad = nº registros del archivo / espacio de direccionamiento

3. Elegir un algoritmo de transformación que obtenga las direcciones base dentro del espacio de direccionamiento que disperse los registros de forma uniforme, a fin de minimizar los sinónimos (si el direccionamiento es a registro) u optimizar su distribución en el caso de direccionamiento a cubo.
4. Utilizar alguna técnica de gestión de desbordamientos. Un desbordamiento se produce cuando la dirección base de un registro o el cubo al que estaba asignado están ocupados; por tanto, debemos pensar en la nueva ubicación de este registro excedentario.

Algoritmo de transformación:

El algoritmo de transformación consta de tres pasos:

1. Si la clave de direccionamiento es alfabética tenemos que convertirla a numérica. Esta modificación se puede realizar multiplicando los códigos ASCII de cada carácter. Este número será, por tanto, la clave direccionamiento.
2. Aplicar una función de transformación a la clave para convertirlo en un número ordinal (*dirección base*) dentro del espacio de direccionamiento.
3. Transformar el ordinal obtenido en el paso anterior en la dirección física que ocupará el registro.

La elección de función de transformación es un factor muy importante para conseguir una dispersión uniforme de las claves, de forma que el número de desbordamientos sea reducido aun con densidades elevadas.

Técnicas de gestión de desbordamientos

Otro aspecto fundamental para conseguir eficiencia en las organizaciones direccionadas dispersas es la gestión de los desbordamientos. Un desbordamiento tiene lugar cuando se produce una colisión y no existe espacio para almacenar el registro sinónimo en el lugar que le corresponde. Este registro se denomina excedentario y será necesario aplicar alguna técnica a fin de ubicarlo en otro lugar dentro del espacio de direccionamiento prefijado o en un área especial (área de desbordamiento).

Cuando utilizamos dispersión con direccionamiento a registro tenemos que siempre que tiene lugar una colisión se produce un desbordamiento. Por el contrario, si el direccionamiento es a cubo, solo existen desbordamientos cuando el bloque se encuentra completo.

Aunque existen diversas técnicas para el tratamiento de los registros excedentarios, sólo veremos, a título de ejemplo, una de las más sencillas, la de **sondeo lineal**, para no extendernos demasiado en este tema⁷.

En la técnica de sondeo lineal, cuando ocurre un desbordamiento se busca, de forma consecutiva, en las posiciones siguientes a la de la dirección base hasta que se encuentra una libre, en la cual se inserta el registro excedentario. Si se llega al final del archivo sin que se haya encontrado una posición vacía, se pasa a la primera dirección del archivo y se sigue el recorrido hasta llegar a la dirección base de la que se partía.

El problema que nos encontramos al utilizar esta técnica es que cuando la densidad del archivo aumenta, el número de accesos, que se han de realizar para localizar un registro que no ocupa su dirección base, aumenta también considerablemente.

5.3. Organizaciones indizadas

Sobre las organizaciones básicas (consecutivas y directas) que acabamos de presentar es posible superponer nuevas estructuras, mediante las cuales se consigue mejorar la eficiencia en el acceso a los registros que cumplan ciertas condiciones de búsqueda; se trata de las organizaciones indizadas.

Puede ocurrir también que la estructura indizada esté unida a los datos, de forma que éstos no tengan una organización básica previa a la construcción del índice.

Una estructura de índice se define sobre uno o más campos de un archivo, llamados **clave de indización**; cada uno de los valores de la clave de indización se asocia a la dirección del registro al que le corresponde dicho valor, véase figura 11.5.

⁷ En las referencias bibliográficas suministradas se pueden encontrar descripciones detalladas sobre la gestión de desbordamientos.

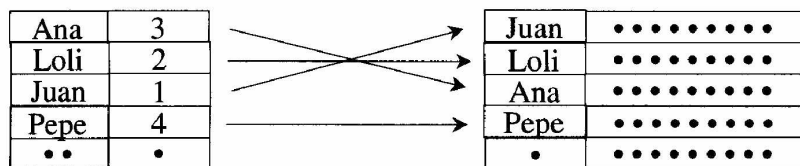


Figura 11.5. Ejemplo de estructura indizada

Un valor de la clave de direccionamiento con la dirección asociada constituye una entrada al índice. El acceso al registro de datos se hace por el valor de la clave de indización, siendo dicha clave el criterio de búsqueda en consultas selectivas. La clave de indización puede ser:

- Un identificador del archivo de datos (índice primario)
- Un conjunto de campos no identificadores (índice secundario)

Sobre un mismo archivo de datos se pueden definir varios índices, cada uno sobre una clave de indización. Cualquier búsqueda selectiva de un registro en una organización indizada obliga a una consulta previa al índice (la cual proporciona la dirección del registro) y un posterior acceso directo al archivo de datos por medio de la dirección obtenida. Por tanto, para optimizar la eficiencia en los accesos, será conveniente mantener el índice, siempre que sea posible, en memoria principal.

Este tipo de organización admite que no exista una entrada al índice por cada registro del archivo (índice no denso), en cuyo caso el archivo de datos tiene que estar ordenado por la clave de direccionamiento, véase figura 11.6. Si existe una entrada por cada registro (índice denso) el archivo no necesita estar ordenado. Los índices no densos admiten la creación de índices sobre los índices, véase figura 11.6, teniendo así estructuras indizadas a varios niveles (índices multinivel).

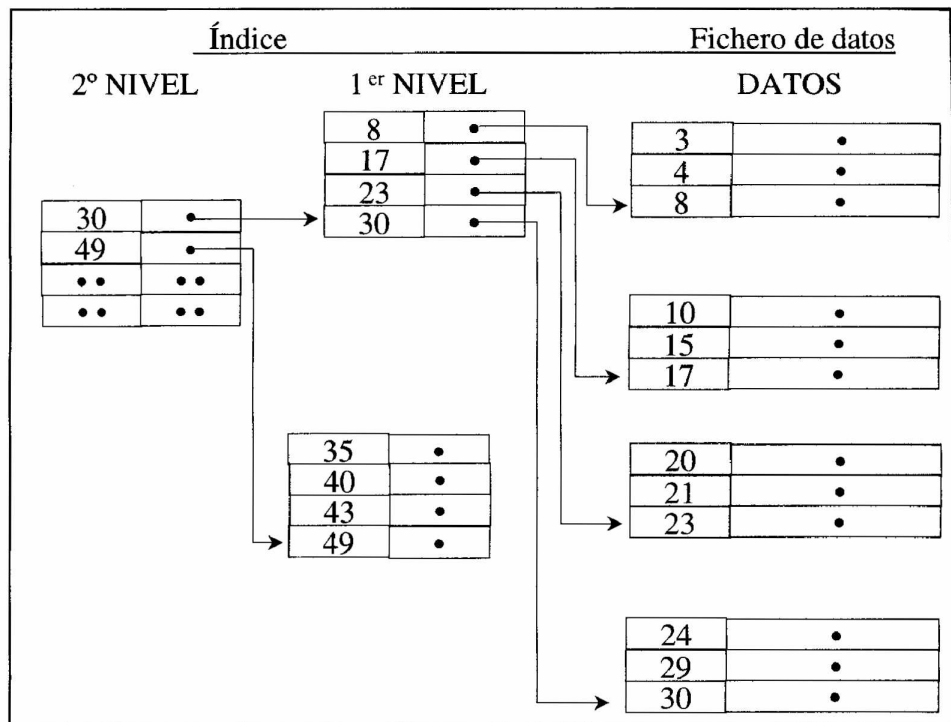


Figura 11.6. Ejemplo de índice no denso multinivel

De entre todas las organizaciones basadas en índices destacan las secuencias indizadas (ISAM), que añaden a las ventajas de las secuenciales (rapidez en el acceso secuencial al archivo), la posibilidad de acceder directamente a ciertos registros (acceso directo) apoyándose en el índice. En este tipo de organizaciones se penalizan las actuaciones respecto a las de tipo secuencial, por la necesidad de actualizar el índice; además al tratarse de una estructura estática, se deterioran cuando se producen muchas actualizaciones.

En las estructuras indizadas, al igual que en las dispersas, se pueden producir desbordamientos en las inserciones, cuando el registro a insertar, o la entrada del índice no tienen espacio en el archivo de datos o de índices respectivamente (los desbordamientos se pueden producir, por tanto, no sólo en el archivo de datos, sino también en el de índices).

Existen diversas técnicas para tratar los desbordamientos parecidas a las de las organizaciones dispersas.

Las técnicas de indización difieren a causa de:

- Tipo de índice (primario/secundario, denso/no denso)
- Estructura del índice (ordenado/desordenado, mononivel/multinivel)
- Organización del archivo de datos (serial/secuencial/disperso)
- Forma de tratar los desbordamientos

Los índices multinivel, a diferencia del ejemplo de la figura 11.6, pueden no tener un número fijo de niveles a fin de evitar desbordamientos y/o reorganizaciones del índice, así como de los datos, cuando se insertan registros; en este caso se fija previamente el número de entradas por cada nivel, y se va incrementando el número de niveles, cuando las inserciones obligan a ello. Surgen así distintos tipos de índices (binarios/equilibrados/etc.), basados, en general, en estructuras de tipo árbol. Entre todas las estructuras en árbol destacan la familia de árboles B, estructuras dinámicas que, a partir de 1972, año en que fueron propuestos por Bayer y McCreight, se extendieron tan rápidamente que, en 1979, Comer afirmaba que se habían convertido en la organización estándar para los índices de un sistema de bases de datos, COMER (1979).

Los nodos del árbol son bloques del espacio de almacenamiento que contienen los valores de las claves de búsqueda y punteros a los nodos hijos; además han de contener, bien los datos, bien los punteros a los mismos. Se asegura una determinada ocupación de los bloques no inferior al 50%, que se estabiliza en un promedio del 69%.

El árbol B es un árbol especial de búsqueda con restricciones adicionales, que garantizan que siempre estará equilibrado (es decir, sus nodos hojas estarán siempre en el mismo nivel) y que su densidad es siempre superior a un determinado valor (el espacio de almacenamiento que se “desperdicia” nunca será excesivo).

Existen diversas variedades de árboles B (B^+ , B^* , con la información asociada o separada de la estructura del árbol, etc.), siendo estas estructuras el fundamento de las organizaciones físicas en los sistemas relacionales. Los árboles B^+ soportan, al igual que las estructuras ISAM, tanto acceso secuencial como directo, con la ventaja adicional de que crecen dinámicamente cuando lo hace la tabla, manteniendo además el orden de la clave de acceso, (si a la tabla no se accede demasiado, puede ser más eficiente emplear la estructura ISAM, ya que dispone de un menor nivel de índices).

Otras técnicas de indización pueden ser híbridas combinando la dispersión con estructuras de índices (a veces llamados índices asociativos), o facilitan el tratamiento de índices o de registros de longitud variable, o bien permiten el manejo de páginas de almacenamiento intermedio mediante árboles B virtuales, etc.

La utilización de índices mejora los tiempos de respuesta ante consultas que impliquen a los atributos indizados, pero disminuye el rendimiento de la base de datos, ya que se debe actualizar el índice cuando se actualizan los atributos sobre los que está definido, además de aumentar el espacio de almacenamiento.

Por estas razones suele ser conveniente indizar la clave primaria (mediante un índice único) en el caso de que el producto no lo haga, las claves alternativas que se utilicen frecuentemente (también mediante un índice único), y aquellas claves ajenas que se utilicen en combinaciones con otras tablas. Sin embargo, en tablas pequeñas, o en aquellas en que prácticamente se recuperan todas las filas, no suele ser conveniente, ya que es mejor una búsqueda secuencial. También se deberá tener en cuenta a la hora de indizar el tipo de datos de los atributos afectados, ya que no es conveniente indizar datos de tipo carácter muy largos.

Recuérdese también que hemos señalado que ante cargas masivas es conveniente crear el índice después de haber insertado los datos.

El SQL-92 y ANSI no especifica ninguna sentencia ni cláusula de diseño físico, ya que su objetivo es el modelo relacional que sólo abarca los niveles lógico global y lógico externo de la arquitectura ANSI/SPARC, véase DE MIGUEL y PIATTINI (1999). Sin embargo, las organizaciones X/OPEN y SQL Access Group han definido una sintaxis para los índices, que es la utilizada para la mayoría de los productos comerciales.

Un ejemplo de definición de índices siguiendo esta sintaxis es el siguiente:

```
CREATE INDEX Ind_Aut ON AUTORES (Institución);
```

En el que el índice de nombre Ind_Aut se define sobre la columna Institución de la tabla Autores.

6. OTRAS TÉCNICAS DE DISEÑO FÍSICO

Existen otras muchas técnicas de diseño físico que permiten al administrador mejorar los tiempos de respuesta y/o el consumo de recursos, entre ellos destacaremos el agrupamiento, la compresión y la redundancia controlada que resumimos a continuación.

6.1. Agrupamientos (cluster) de tablas

Algunos sistemas permiten agrupar tablas cuyas filas comparten un grupo de atributos denominados clave de agrupamiento. Como se muestra en la figura 11.7, esta técnica realmente proporciona una desnormalización física de las tablas, ya que éstas

se encuentran físicamente agrupadas. Sin embargo, lógicamente siguen siendo dos tablas independientes, por lo que el agrupamiento resulta transparente al usuario.

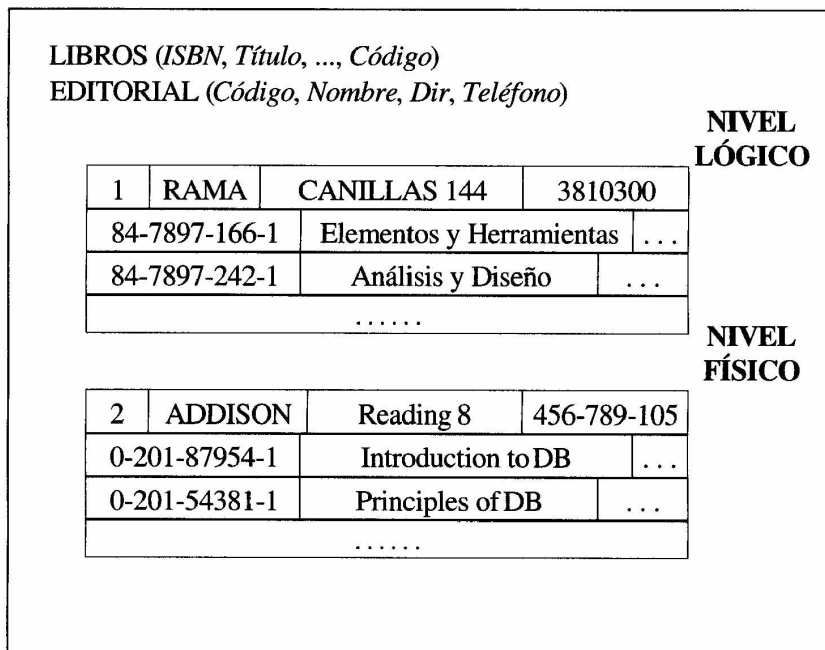


Figura 11.7. Ejemplo de agrupamiento de tablas.

Con este tipo de mecanismos se consigue mejorar de forma considerable los tiempos de respuesta en la combinación de tablas, pero empeoran los recorridos completos de las tablas por separado, así como las actualizaciones de tablas que se encuentran en varios agrupamientos.

Para crear un agrupamiento como el que se muestra en la figura 11.7, se utilizan (en ORACLE) las siguientes sentencias:

```
CREATE CLUSTER Lib_Edi (Código NUMBER (5));
```

```
CREATE TABLE Editorial
(Código NUMBER (5) PRIMARY KEY,
...)
CLUSTER Lib_Edi (Código);
```

```
CREATE TABLE Libros
(Isbn NUMBER (10) PRIMARY KEY,
```

...
Codigo NUMBER (5) FOREIGN KEY REFERENCES Código Editorial)
CLUSTER Lib_Edi (Código);

Hay que destacar que tanto para los índices del apartado anterior como para los agrupamientos, también es posible, en algunos productos, especificar las características de los bloques físicos.

6.2. Técnicas de compresión

Otra técnica a tener en cuenta es la compresión de datos, que si bien permite reducir el espacio requerido para almacenar los datos (disminuyendo, por tanto el número de operaciones de entrada/salida a disco), requiere más proceso debido a la necesidad de descomprimir los datos que se recuperan.

La técnica más utilizada es la compresión diferencial, en la que en lugar de almacenar el valor de un atributo, se almacena la diferencia entre cada valor y el que le precede.

También existe lo que se denomina compresión jerárquica, DATE (1995), en los agrupamientos, en la cual los valores de la clave por la que se agrupan las filas se almacenan una sola vez.

A continuación se muestra un ejemplo (en el sistema INGRES) en el que la tabla AUTOR se modifica para que posea una estructura de almacenamiento en forma de árbol B, con compresión de datos pero no de la clave:

MODIFY Autores TO BTREE ON Código
WITH COMPRESSION = (NOKEY; DATA);

6.3. Redundancia de datos

También hay que considerar la posibilidad de duplicar ciertos datos de una tabla en otra o de almacenar atributos derivados, con el fin de evitar accesos a tablas consultadas frecuentemente. Ahora bien, se debe siempre garantizar la consistencia de la base de datos, por lo que esta redundancia deberá ser controlada por el sistema, pudiéndose utilizar para ello disparadores.

En resumen, las organizaciones físicas de las bases de datos se han ido complicando en el transcurso del tiempo a fin de mejorar la eficiencia de los sistemas; el administrador de la base de datos ha de conocer bien las características del producto que está utilizando a fin de elegir aquellas opciones que optimicen recursos y tiempos de respuesta.

KVS@C

- 12. Herramientas de desarrollo:
Lenguajes de cuarta generación**
- 13. Sistemas de diccionarios de
recursos de información**
- 14. Herramientas CASE y diseño
de bases de datos**

*P
A
R
T
E*

IV

HERRAMIENTAS DE DESARROLLO: LENGUAJES DE CUARTA GENERACIÓN

El proceso evolutivo de los SGBD ha tenido una considerable influencia en la aparición y evolución de herramientas de desarrollo, entre ellas, los lenguajes de cuarta generación (L4G). En este capítulo empezaremos presentando una rápida visión de la evolución de los lenguajes hasta llegar a los L4G¹, analizando sus ventajas e inconvenientes y describiendo brevemente las características de éstos y de otros tipos de lenguajes, como orientados a objetos y visuales.

1. EVOLUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

El análisis de la evolución de los lenguajes de programación tiene interés para comprender la características de las herramientas de desarrollo de cuarta generación, en especial, de este tipo de lenguajes, por lo que incluimos un resumen de dicha evolución basado en PIATTINI *et al.* (1996).

1.1. Lenguajes de primera y segunda generación

Aunque sea sólo por motivos históricos, es necesario incluir en la evolución de los lenguajes de programación los lenguajes de primera generación (código máquina) y de segunda generación (ensamblador) que se consideran aplicables a aquellas partes

¹ 4GL en inglés

de un sistema que requieren una gran velocidad, pero cuya utilización no se encuentra en la actualidad demasiado extendida.

Al utilizar un lenguaje ensamblador, el código fuente se ensambla, línea a línea, en código máquina, informando al programador de si existe algún problema con dicho código. Además, los lenguajes de segunda generación permiten combinar diferentes archivos de código fuente en un solo programa ejecutable utilizando un "enlazador" (*linker*).

1.2. Lenguajes de tercera generación

Debido a la espectacular evolución de los rendimientos del hardware respecto a su precio, los lenguajes de segunda generación perdieron su protagonismo en función de lenguajes de alto nivel como COBOL, C, PASCAL, FORTRAN, etc. En estos lenguajes, que son más comprensibles para el programador, una sentencia se traduce en varias instrucciones de máquina.

Con esta generación empiezan a aparecer realmente los entornos de programación, que permiten compilar y depurar programas, incluso en máquinas distintas de la de destino.

En general podemos distinguir dos tipos de traducción del código fuente a código máquina:

- De una sola vez, como una tarea preparatoria llevada a cabo por un compilador.
- En cada ejecución, por un intérprete, el cual va traduciendo cada sentencia del programa fuente a código de máquina, ejecutándolo a continuación y advirtiéndole si encuentra algún error, en cuyo caso permite la corrección del mismo y la continuación del proceso.

La compilación y el enlace pueden llegar a ser lentos, como consecuencia de la corrección de los errores introducidos en el código, pero una vez que se ha compilado con éxito, el programa es más rápido y presenta un código más compacto que un intérprete.

En un compilador podemos distinguir:

- Un analizador sintáctico.
- Un analizador semántico.
- Un optimizador.
- **Traductores de código fuente a lenguaje máquina.**

Un intérprete, por su parte, suele aparecer integrado con un editor del lenguaje así como con un generador de código máquina.

13. Lenguajes de cuarta generación

Aunque existen diversas definiciones de lenguaje de cuarta generación (L4G) y más de trescientos productos que se engloban en esta categoría, todas ellas coinciden en que un L4G se caracteriza por ser *no-procedimental*, lo que permite al usuario preocuparse de "QUÉ" sin necesidad de especificar "CÓMO" hacerlo. En definitiva se trata de especificar el resultado deseado más que las acciones necesarias para obtener el resultado.

Tradicionalmente se los clasifica en dos grandes grupos:

- Lenguajes para usuario final, lo que se ha denominado centro de información o infocentro (*information center*), que nacieron con la idea de independizar a los usuarios del control del departamento de informática. Se centran sobre todo en la facilidad de uso y flexibilidad.
- Lenguajes para el informático (analista/programador), es decir, para el centro de desarrollo (*development center*), cuyo fin es facilitar el desarrollo de aplicaciones sofisticadas y permitir la construcción rápida de aplicaciones por medio del prototipado.

Este tipo de lenguajes surge al extenderse la utilización de los SGBD; como señala COBB (1985), "los SGBD han desempeñado un importante papel en el proceso evolutivo de los lenguajes informáticos. De hecho, sin la independencia de los datos que ofrecen los SGBD, los lenguajes de cuarta generación nunca habrían existido".

La última evolución de los L4G en este sentido, es la aparición de lenguajes independientes del SGBD, que ofrecen una gran flexibilidad, permitiendo elegir el SGBD que mejor se adapte a cada caso. Sobre todo resultan útiles para fabricantes de aplicaciones que tienen que ejecutarse sobre diferentes SGBD.

14. Lenguajes de quinta generación

A veces se emplea el término *lenguajes de quinta generación* (L5G) para englobar los lenguajes que vienen apareciendo desde finales de los años ochenta tales como:

- Lenguajes lógicos
- Lenguajes funcionales
- Lenguajes paralelos

que se caracterizan por su especialización en ciertas áreas de aplicación.

1.5. Lenguajes orientados a objetos

Aunque algunos expertos los consideran como L5G, la historia de los lenguajes orientados al objeto empezó a finales de la década de los sesenta con el lenguaje Simula, diseñado en el *Norwegian Computing Centre* para modelar y simular sistemas. Por otro lado, a principios de los setenta en el *Xerox Palo Alto Research Center* se desarrolló, dentro de las investigaciones orientadas al sistema Dynabook (computador personal orientado a gráficos), el lenguaje Smalltalk.

A principios de los ochenta se empezó a trabajar en extensiones del lenguaje C (C++ y Objective-C), de importancia creciente debido a la difusión de los entornos UNIX.

A mediados de los ochenta apareció otro lenguaje importante en esta área, Eiffel, así como enfoques híbridos del tipo Object Pascal. A mediados de los noventa se han desarrollado extensiones orientadas al objeto de COBOL (OOCOBOL) y de Ada (Ada-95), a la vez que aparecen nuevos lenguajes como Dylan o Java. Hay que destacar el gran impacto de este último, debido a la explosión del fenómeno Internet.

También en la década de los noventa se modifican los L4G existentes, a los que se les incorporan varios de los principios del paradigma de la orientación a objetos, como pueden ser el encapsulamiento, la generalización o el polimorfismo.

Los entornos orientados al objeto se caracterizan por poseer, además de todos los elementos de los lenguajes de tercera generación, un *visualizador* o *inspector* (*browser*), que permite recorrer las diferentes jerarquías de objetos de las aplicaciones o bibliotecas.

1.6. Lenguajes "visuales"

Dentro de este apartado se pueden distinguir dos grandes grupos, BURNETT *et al.* (1997):

- Lenguajes de programación visual propiamente dichos, son aquellos que tienen una "sintaxis visual", esto es, que algunos de los terminales de su gramática son gráficos (imágenes, formas, animaciones, etc.) e incorporan información espacial (intersección, contenido, etc.) y atributos visuales (color, alineación, etc.). Ejemplos de este tipo son HI-VISUAL, ObjectWorld, etc.
- Entornos visuales, son herramientas gráficas que se utilizan para crear, modificar y examinar programas (con ayuda del ratón) escritos en un lenguaje de programación textual, como puede ser el caso de VisualBasic.

Recientemente empiezan a aparecer lenguajes que combinan las facilidades visuales con la tecnología de objetos, y que se conocen por las siglas VOOPL (Visual Object Oriented Programming Language), véase BURNETT *et al.* (1997).

Hay que destacar que la facilidad de aprendizaje de los lenguajes visuales han determinado su acogida por los usuarios y su correspondiente difusión. Como señala LEWIS (1996), si se comparan los ratios de las curvas de aprendizaje de estos lenguajes con los requeridos por los Sistemas de Gestión de Bases de Objetos (SGB0), los lenguajes tradicionales y las herramientas CASE, podemos entender la gran difusión de los primeros. También puede explicarse de esta manera parte del fracaso que tuvieron las herramientas CASE, debido a que presentan curvas de aprendizaje demasiado elevadas.

2. COMPONENTES DE UN L4G

Como ya hemos señalado la variedad de L4G es muy considerable, ya que no existe un modelo de referencia o estándar que especifique los componentes que debe poseer. A pesar de ello, típicamente podemos encontrar en un entorno de este tipo:

- **Diccionario**, es el núcleo del sistema que almacena información sobre las aplicaciones que se construyen (definición de campos, pantallas, informes, etc.), por lo que sirve de fuente de documentación del sistema. Además proporciona valores por defecto que permiten al usuario ahorrar tiempo y esfuerzo, reduciendo la cantidad de información que debe suministrar.
- **Entorno de desarrollo**, compuesto de compilador y/o intérprete. En este último caso suele ser interesante que el sistema permita, apoyándose en los valores del diccionario de datos, controlar el comportamiento de las aplicaciones en tiempo de ejecución sin tener que recompilar ante modificaciones.
- **Diseñador de pantallas y menús**, que ofrece una serie de facilidades para que el usuario pueda directamente (de forma gráfica) especificar los formatos de pantalla, las posiciones de cada campo, etc. Suele complementarse con herramientas especializadas en gráficos en 2 y 3 dimensiones (tartas, barras, etc.).
- **Generador de informes**, que permite especificar las cabeceras, pies, rupturas, cálculos, etc.
- **Interfaces**, con distintos elementos del entorno, que permitan, por ejemplo, el intercambio de datos entre estaciones de trabajo y computadores principales (*mainframe*)

- **Plantillas** (*templates*) pre-elaboradas que permiten llevar a cabo ciertas operaciones, como leer un registro de un archivo, liberando al desarrollador de tareas repetitivas.
- **Componentes** (ventanas de diverso tipo, botones, etc.) que se pueden incluir en las aplicaciones con sólo copiarlas con la ayuda del ratón.

3. VENTAJAS E INCONVENIENTES DE LOS L4G

Los lenguajes de cuarta generación presentan una serie de características diferenciadoras respecto a los de generaciones anteriores. A continuación señalamos las principales ventajas e inconvenientes que, en general, presenta esta tecnología, aunque debe tenerse en cuenta que es necesario realizar un análisis detallado de cada lenguaje y entorno en particular.

3.1. Ventajas

Un aspecto común a todos los entornos de cuarta generación es que son sistemas cada vez más "amistosos" (*user-friendly*) fáciles de usar, para lo cual se intenta colmar el vacío existente entre la forma de descripción del sistema inteligible para el usuario final y la manera en que lo describe el informático. Así, por ejemplo, los L4G facilitan la navegación por la base de datos evitando que el usuario tenga que hacer las combinaciones (*join*) entre tablas.

Este aspecto va relacionado con la mejora de la productividad respecto a lenguajes tradicionales como COBOL. El aumento de productividad, que en los folletos de márketing de algunos productos se eleva hasta el 3000%, podemos situarlo de forma realista en una proporción de 5 a 1. Hay que recordar, sin embargo, que esta ganancia en productividad se consigue a costa de eficiencia de procesamiento; teniendo en cuenta el continuo cambio que se observa desde hace años en la proporción del coste del equipo físico (*hardware*) y sus prestaciones en comparación al soporte lógico (*software*), resulta evidente que se favorece cada vez más el ahorro de "ciclos de memoria humana" en lugar de "ciclos de memoria del computador".

Otro importante aspecto de los L4G es facilitar el desarrollo mediante "maquetas", que permiten disminuir la desconfianza del usuario, hacer más fluida la comunicación entre el usuario y el desarrollador, contrastar creencias, confirmar la factibilidad de ciertos diseños, etc.

Incluso, la mayor parte de los L4G permiten crear verdaderos "prototipos", que transforman el ciclo de vida de las aplicaciones, adoptándose un enfoque evolutivo e incremental; lo que tiene grandes efectos positivos tanto para el producto software

como para el proceso de desarrollo, y que ha tenido gran éxito en diferentes situaciones, véanse, por ejemplo, GORDON y BIEMAN (1994).

Los entornos de cuarta generación ofrecen bastante funcionalidad "por defecto", que se puede incorporar en los sistemas que hay que desarrollar, lo cual no sólo mejora, como hemos señalado, la productividad, sino que también disminuye el número de errores al acceder a archivos y al manipular pantallas.

Una última ventaja que queremos destacar es la facilidad de construir aplicaciones portables y con grandes posibilidades de adaptación (*customización*) que suelen ofrecer los L4G.

3.2. Inconvenientes

A pesar de todas las ventajas enumeradas en el apartado anterior, cabe destacar también algunos inconvenientes que presentan todavía los L4G.

Por lo que respecta al diccionario, éste suele ser propietario y no permite una fácil integración con otros componentes desarrollados en diferentes entornos. De hecho, es importante considerar la necesidad de integrar y coordinar el conjunto de herramientas que forman el entorno de cuarta generación, que debe permitir comunicar los resultados de una con otra sin que se tenga que invertir tiempo en este tipo de tareas. Un problema añadido en este sentido es la falta de mecanismos de sincronización automáticos entre el catálogo de los SGBD, los diccionarios de los L4G y los repositorios de herramientas CASE (véase capítulo siguiente).

Ya señalamos el problema de la eficiencia, que no sólo viene condicionado por el propio entorno (que suele consumir gran cantidad de recursos de máquina), sino que depende muchas veces de que el L4G "conozca" como funciona el optimizador del SGBD para que pueda mejorar la velocidad de acceso.

En lo que respecta al desarrollo mediante prototipos, muchos L4G se prestan a un desarrollo "rápido y sucio" (*quick & dirty*), por lo que evidentemente se resiente la calidad de las aplicaciones. A esto se añade la falta de buenas facilidades de depuración, que hacen que la etapa de prueba en los entornos de cuarta generación consuma mucho tiempo y muchas veces no sea tan clara como en los L3G, en los que se puede seguir más fácilmente el código. Por tanto, la utilización de L4G no debe evitar utilizar métodos de análisis y diseño rigurosos, así como técnicas de aseguramiento de calidad, seguridad y auditoría, PIATTINI (1992).

Otra desventaja de algunos L4G es su falta de flexibilidad a la hora de abordar algunas tareas de bajo nivel, lo que puede hacer necesario la utilización de lenguajes de tercera generación (como el lenguaje C) para construir una aplicación. En efecto,

los L4G están mejor adaptados a aplicaciones intensivas en datos en las que se realizan operaciones sencillas sobre grandes volúmenes de datos, que a aplicaciones de proceso complejo sobre pocos datos. Muchos L4G ofrecen para este último caso salidas a código en L3G, aunque lo ideal es no tener que salir al L3G sino que el L4G sea completo en cuanto a operadores, gestión de errores, funciones de bajo nivel, etc.

Un aspecto poco destacado, y sin embargo de gran importancia, como señala LEBER (1991), es que al pasar de un L3G tradicional a un L4G basado en SQL hay que enfrentarse a la lógica trivaluada, véase DE MIGUEL y PIATTINI (1999).

Por último, cabe destacar que los L4G desdibujan la frontera todavía existente en muchas organizaciones entre analista y programador, cobrando mayor importancia la existencia de analistas/programadores que diseñan las aplicaciones y las implementan con ayuda de un entorno de herramientas integradas.

SISTEMAS DE DICCIONARIOS DE RECURSOS DE INFORMACIÓN

Para una efectiva gestión de la información en las organizaciones es imprescindible disponer de arquitecturas y de herramientas que faciliten la integración; el núcleo de todas ellas es el *Diccionario de Recursos de Información* (DRI), en el que se almacena el conocimiento de los usuarios acerca de los datos. En este capítulo se analizan el concepto de DRI y la evolución histórica de los instrumentos que han ido almacenando las descripciones de los datos hasta llegar al DRI. También se presenta el entorno en el que se desenvuelven estos sistemas y el papel que desempeñan en la empresa, se analiza su contenido y se estudian una serie de estándares relativos a los DRI.

1. INTRODUCCIÓN

A medida que se incrementa la demanda de información, se eleva la prioridad de la gestión efectiva de los recursos informativos de las empresas, por lo que surge la necesidad de instrumentos capaces de ayudar a los diseñadores de la base de datos y a los distintos usuarios a conocer y gestionar el contenido informativo de la misma.

A menudo, los usuarios del SI se ven sumergidos en grandes volúmenes de datos cuya utilidad se ve muy disminuida al no estar debidamente documentados y al no tener asociada la semántica precisa para conseguir una información verdaderamente útil y valiosa. En muchas empresas existen lo que podríamos llamar *islas de datos* que, al tratarse de datos no integrados, disminuyen el valor de la información de la que pueden disponer los usuarios.

De lo que acabamos de exponer se deduce la necesidad de que exista una verdadera arquitectura de datos que facilite la integración. El núcleo de dicha arquitectura será el *Diccionario de Recursos de Información* (DRI), que contendrá las descripciones de los datos que constituyen el SI y alrededor del cual se irán situando todos los elementos del mismo.

Los recursos informativos de las instituciones se gestionan almacenando, administrando y controlando los denominados *metadatos*, esto es, los datos que definen y describen los datos de la misma. A lo largo de la historia de la informática han ido apareciendo *almacenes* de metadatos que han recibido distintos nombres, siendo los términos más difundidos en el entorno de los SGBD los de **Catálogo** y **diccionario de datos** (DD) y, en el ámbito de las herramientas CASE, el de **repositorio**.

2. EVOLUCIÓN HISTÓRICA: DE LOS DIRECTORIOS/ DICCIONARIOS DE DATOS AL DICCIONARIO DE RECURSOS DE INFORMACIÓN

Desde el comienzo de la informática se ha intentado plasmar en los sistemas de información el *conocimiento* que los usuarios poseían sobre su dominio de aplicación, almacenando datos relativos al mismo. En las primeras etapas, los sistemas de información estaban soportados básicamente por un conjunto de programas en los que se *embebía* la descripción de los datos, así como algunas de sus características y también, aunque en una mínima parte, sus restricciones, es decir, su **semántica**.

Estos sistemas *orientados al proceso* resultan, muy difíciles de mantener y demasiado complejos, por lo que se plantea la necesidad de centralizar las descripciones de los datos para conseguir sistemas más coherentes, eficientes y adaptables a los cambios. A fin de atender estos objetivos surgen los SGBD, DE MIGUEL y PIATTINI (1999).

Como un componente del SGBD aparece el **directorio de datos**, que es el encargado de describir dónde y cómo se almacenan los datos de la base, el modo de acceso y otras características físicas de los mismos, atendiendo de este modo las peticiones de los programas y de los procesos. Un directorio de datos contiene, en definitiva, las especificaciones necesarias para pasar de la representación externa de los datos a la representación interna de los mismos, y ha de estar siempre en un formato legible por la máquina; su objetivo principal es transmitir al SGBD la información necesaria para poder acceder a los datos contenidos en la base.

Con independencia de los directorios surgen los llamados **diccionarios de datos**, donde se reúne la *información sobre los datos* almacenados (descripciones —narrativas y técnicas—, estructuras, consideraciones de seguridad, edición y usos de las aplicaciones de los mismos, etc.) que los usuarios necesitan para comprender el significado, esto es, el aspecto lógico de los datos. Al principio, estos diccionarios eran simplemente listas

manuales elaboradas por los usuarios o manejadas por un conjunto de programas *hechos a medida* por el administrador de la base.

La complejidad de los diccionarios de datos se fue haciendo cada vez mayor, expandiendo el número de funciones que se le asignaron originalmente, así como el entorno operativo en el que se utilizaban. En los años setenta aparecen varios paquetes software de este tipo: DB/DC DATADictionary, DATAMANAGER, DATADictionary SYSTEM, ADR/DATADictionary, LEXICON, etc.

Algunas veces estos paquetes llevan a cabo tanto las funciones de diccionario como las de directorio, denominándose entonces: **diccionario/directorio de datos (DD/D)**.

En la figura 13.1 se presentan dos posibles arquitecturas para los SI que posean SGBD, diccionario y directorio. En la primera, el diccionario y el directorio son dos elementos distintos sin ninguna interfaz directa. El usuario puede alimentar el diccionario con datos sobre su sistema —sobre todo con fines de documentación—, pero tiene, además, que indicar al SGBD los elementos a crear en la base, a través de un LDD (Lenguaje de Definición de Datos), almacenándose estas definiciones en el directorio. Este tipo de diccionarios que no tienen una relación directa con el SGBD se denominan diccionarios pasivos.

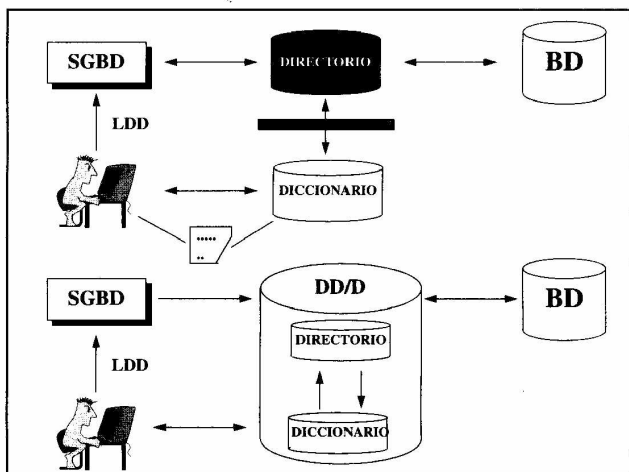


Figura 13.1. Posibles arquitecturas de los SI apoyados en diccionarios y directorios

En la segunda arquitectura, el diccionario y el directorio están integrados en un DD/D, que sirve tanto a los usuarios como al propio SGBD, tratándose por tanto de un diccionario activo.

Por otro lado, la creciente complejidad de la concepción y diseño de los SI ha llevado a la construcción y comercialización de herramientas (véase capítulo 14)

conocidas bajo el nombre genérico de CASE (Computer Aided Software/System Engineering), que contienen un diccionario llamado **enciclopedia** o **repositorio**, donde se almacena los datos generados durante el ciclo de vida de un SI: esquemas, grafos, matrices, información relativa a la gestión de proyectos, gestión de configuraciones, etc. Aunque este repositorio no suele ser activo, en algunos casos la herramienta CASE facilita instrumentos para cargar directamente las descripciones de los datos obtenidas en la etapa de diseño, en los catálogos propios de los SGBD más extendidos.

También los Lenguajes de 4ª Generación (L4G) suelen tener, como vimos en el capítulo anterior, su propio diccionario donde se almacenan definiciones y otras características de los datos necesarias para el funcionamiento de estos lenguajes.

Hace algunos años ha aparecido un nuevo concepto, el de **diccionario de recursos de información** (DRI)¹, que pretende ser el eslabón final en la evolución de los *almacenes de datos*. El DRI constituye el depósito integrado de toda la información acerca de los datos sobre la organización, automatizados o no, que son utilizados para efectuar las labores de planificación, control y operación y que permitan a la empresa cumplir sus objetivos. Los DRI engloban, de algún modo, las capacidades y funciones de todos los *almacenes de datos* anteriores.

En la figura 13.2 se presentan de forma gráfica, los distintos elementos que hemos analizado tal como aquí se han expuesto en un intento por precisar y clarificar unos conceptos respecto a los cuales no existe en la actualidad consenso y cuya terminología, resulta, a veces, bastante confusa².

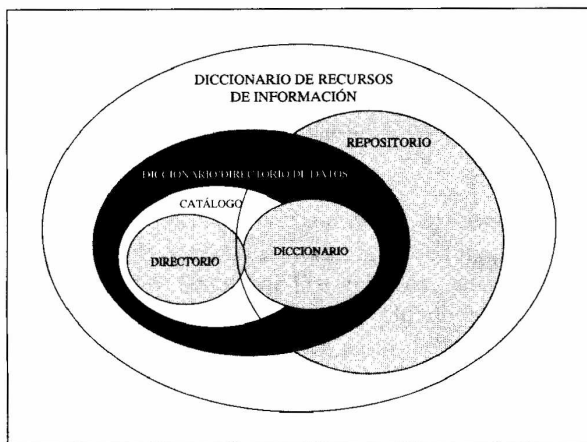


Figura 13.2. Distintos almacenes de datos

¹ Las siglas inglesas son IRD (*Information Resource Dictionary*), que son las utilizadas por los organismos de estandarización ANSI e ISO.

² De hecho, algunos expertos denominan repositorio a lo que hemos llamado DRI, dejando este último término para los estándares definidos por ISO o por ANSI.

Con independencia de la terminología empleada, es importante observar la existencia de dos tipos de *repositorios*, como señala YIN (1994):

- repositorios con "r" (minúscula), que almacenan objetos para una herramienta CASE en particular.
- Repositorios con "R" (mayúscula), que tienen un alcance mayor, se basan en estándares e instrumentan un modelo de información abierto y extensible, soportando un entorno integrado de ingeniería de software, y que coincide, en gran parte, con lo que nosotros hemos llamado DRI.

Mientras los primeros constituyen el núcleo de una herramienta CASE, los segundos pretenden ser el corazón de toda la arquitectura de información de la empresa, **sirviendo de soporte para la integración de sistemas**.

Esto se consigue, en parte, gracias a que el diccionario es capaz de proveer independencia de metadatos para múltiples entornos, y en las empresas de un cierto tamaño es usual encontrar distintos entornos hardware/software con los que tiene que interactuar el SI. El diccionario describe los datos de manera independiente del soporte físico, facilitando así la migración de aplicaciones y rentabilizando el esfuerzo de diseño y concepción de sistemas y, sobre todo, consolidando los recursos informativos de las empresas.

Por lo que se refiere al concepto de *catálogo* de un SGBD, que surge con los SGBD relacionales, hay que destacar que se podría considerar como un diccionario/directorio de datos pero de bajo contenido semántico. Un aspecto importante del mismo es que mientras que en la mayoría de los casos el SDRI es un sistema de gestión propio del DRI e independiente del SGBD —véase figura 13.3a—, en el caso de los catálogos el sistema de gestión es el propio SGBD, y el DRI una base de datos *más*, por lo que se denomina, a veces, *metabase*, véase figura 13.3b.

Este último enfoque, el de la metabase, no cabe duda que es mucho más conveniente para los usuarios y, en particular, para el administrador; ya que accederemos a la metabase con el mismo lenguaje que a las demás bases de datos (por ejemplo, SQL en los SGBD relacionales) y será mucho más fácil de extender que un diccionario separado de las otras bases de datos. Tiene, sin embargo, el inconveniente de que suele estar limitada a describir únicamente los elementos contenidos en las bases de datos y, además, que su capacidad para recoger la semántica suele ser reducida.

Según CODD (1990) uno de los requisitos para que un SGBD pueda considerarse verdaderamente relacional es que "soporte un catálogo dinámico en línea en el que la descripción de la base de datos se represente como cualquier otro dato, permitiendo a los usuarios autorizados aplicar el mismo lenguaje relacional tanto a la descripción de la base de datos como a los datos regulares".

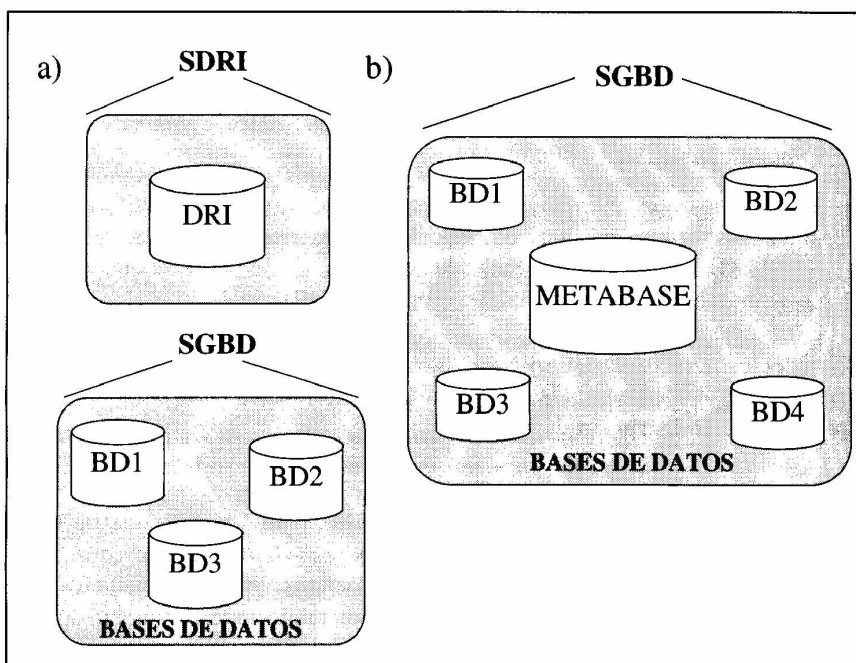


Figura 13.3. SDRI y SGBD independientes vs. metabase

3. EL SDRI Y SU ENTORNO

En la figura 13.4 se presenta la arquitectura de un SI compuesto por el DRI y el software encargado de gestionarlo —SGDRI (Sistema de Gestión de Diccionario de Recursos de Información³)— como eje alrededor del cual se sitúa el resto de elementos del sistema: programas, lenguajes de cuarta generación, generadores de informes y de pantallas, herramientas CASE y facilidades de usuario. Se observa también, en la figura, que un SDRI se puede apoyar en un SGBD, lo que permite aprovechar muchas de las funcionalidades que éste proporciona, o bien directamente en un Sistema de Gestión de Archivos.

En el diccionario se almacenarán, incluso, la sintaxis y semántica de los diferentes lenguajes que soporten los componentes del DRI, para poder producir, así, de forma automática las definiciones de datos necesarias.

³ Denominamos **SDRI** (*Sistema de Diccionario de Recursos de Información*) al conjunto formado por el DRI y el SGDRI.

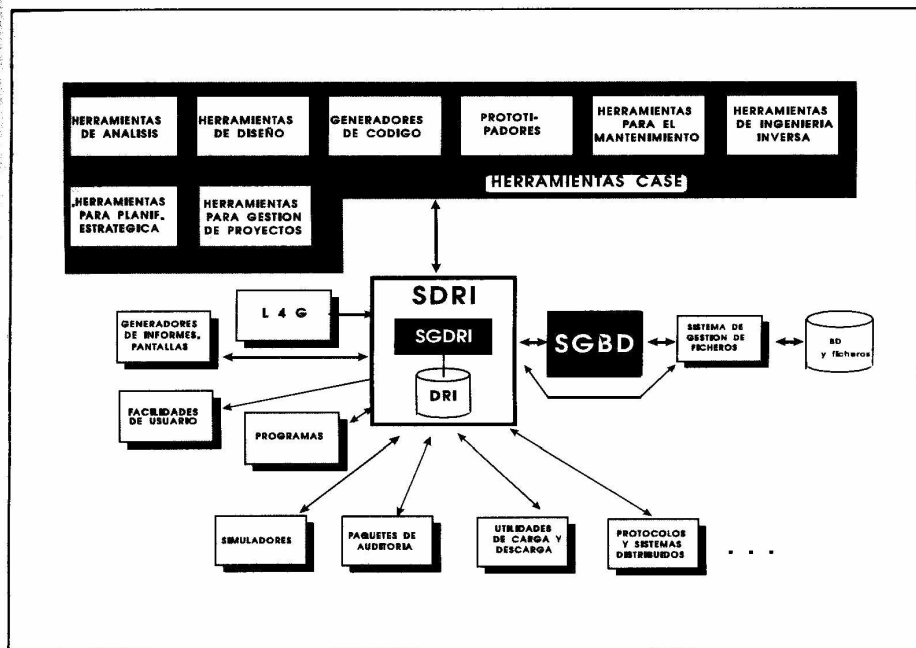


Figura 13.4. El SDRI y su entorno

4. PAPEL DEL DICCIONARIO EN LA EMPRESA

Como hemos señalado, el diccionario es un elemento clave en la gestión de recursos de información, ya que proporciona una fuente de información integrada, aumentando así la disponibilidad global de los recursos; como señala HODGES (1993): *"el reto de integrar diversos datos y las aplicaciones que los manipulan en una base de información significativa es la principal razón para los sistemas de repositorio"*.

En NARAYAN (1988) se analizan los diferentes aspectos en los que un diccionario puede resultar útil para la gestión de recursos de información:

- **Planificación estratégica**, ya que el diccionario permite identificar los principales componentes (tanto de procesos como de datos) del sistema de información y del *negocio* de la empresa.
- **Administración de la base de datos**, debido a la documentación que posee el diccionario sobre los datos, sobre la relación entre datos y aplicaciones y sobre procedimientos de recuperación, estadísticas, etc. que facilitan la labor del administrador de la base de datos.

- **Desarrollo de sistemas**, ya que el diccionario se emplea para almacenar la gran cantidad de documentación generada durante todo el ciclo de vida y sirve también para automatizar ciertas actividades de las distintas fases del desarrollo.
- **Explotación**, no sólo a nivel de documentación, sino que el propio diccionario podría disparar la ejecución de trabajos, incluyéndole la información de planificación adecuada.
- **Seguridad**, ya que podría utilizarse el diccionario para verificar los accesos a los componentes del sistema, teniendo en cuenta las características de éstos y los privilegios de los usuarios.
- Informática de **usuario final**, ya que si el diccionario presenta interfaces amistosas, podría ayudar a los usuarios finales a localizar la información requerida.

El diccionario desempeña, además, un papel primordial en la interoperabilidad de sistemas, sirviendo de soporte para el funcionamiento de sistemas de información en entornos distribuidos, homogéneos o heterogéneos.

5. CONTENIDO DEL DICCIONARIO DE RECURSOS DE INFORMACIÓN

Para cumplir sus objetivos, un DRI debe contener, entre otras, información relativa a las siguientes áreas:

- Organización a la que va a servir, los *modelos* de empresa, *reglas de negocio* (*business rules*), véase PIATTINI (1994), recogiendo las características más significativas de los distintos sistemas que integran la organización y del entorno de la misma.
- Usuarios de la organización y del sistema de información.
- Subsistemas que integran cada uno de los sistemas considerados.
- Componentes físicos y elementos de entorno que integran cada subsistema.
- Estructuras de datos, archivos y BD de cada uno de ellos, así como las interrelaciones existentes entre estos datos y las reglas de integridad.
- Seguridad y confidencialidad de los datos.
- Programas, aplicaciones o rutinas de cada subsistema.
- Metodologías a emplear.

También puede contener información sobre el propio diccionario, su estructura y características, esto es, *meta-metadatos*.

Toda esta información se suele representar siguiendo un modelo conceptual (modelo E/R u otro con mayor capacidad semántica), por medio de entidades (por ejemplo: PROGRAMA, USUARIO, REGISTRO, PANTALLA, etc.), interrelaciones entre dichas entidades (UTILIZA, CONTIENE A, etc.) y atributos de las entidades y de las interrelaciones (FECHA DE CREACIÓN, CÓDIGO, NÚMERO DE LÍNEAS, etc.).

Los diccionarios deben ser extensibles (para poder adaptarlos al cliente), de modo que cada usuario pueda definir nuevos objetos, utilizando tanto éstos como los facilitados de forma estándar por el sistema; es, por tanto, interesante que el diccionario se presente en un formato *abierto*, esto es, que sea accesible a través de otras herramientas.

6. ESTÁNDARES SOBRE SDRI

La necesidad de resolver los problemas de integración, interoperabilidad, gestión distribuida de los recursos de información, etc. constituye una motivación para la definición de estándares que faciliten la cooperación de los sistemas de información en entornos abiertos.

Existen en la actualidad varios estándares y normas propuestos para los SDRI o temas relacionados con éstos —véase figura 13.5—.

PROMOTOR NIVEL	ORGANISMO OFICIAL	GRUPO DE INTERES	INDUSTRIA
INTERNACIONAL	ISO/IEC IRDS	EIA CDIF	IBM REPOSITORY DIGITAL ATIS
REGIONAL		ECMA PCTE	
NACIONAL	ANSI IRDS FIPS IRDS		

Figura 13.5. Principales estándares para SDRI

En primer lugar cabe destacar el elaborado por el *Institute for Computer Sciences and Technology* del organismo estadounidense National Bureau of Standards, que ha sido aceptado por el comité X3H4 de ANSI (American National Standard Institute), y que sirve como base para un estándar FIPS (Federal Information Processing Standard). Este último fue elaborado originariamente por el *Institute for Computer Sciences and*

Technology del National Bureau of Standards (NBS), y adoptado por el comité X3H4 de ANSI, quien finalmente lo aprobaría en octubre de 1988. En abril de 1989, el NIST (National Institute of Science and Technology, nombre que adoptó el NBS), aprobó este estándar como FIPS (Federal Information Processing Standard).

Recientemente han aparecido en el mercado norteamericano algunos productos que siguen el estándar ANSI; sin embargo, dicho estándar no ha terminado de cuajar en el entorno empresarial

Otro estándar "*de iure*" ha sido el propuesto por el grupo de trabajo SC21/WG3 de ISO/IEC JTC1. En estos momentos el JTC1 está reorganizando sus actividades y cuestionándose la continuación de estos estándares.

El anuncio hecho por IBM en septiembre de 1989 relativo a AD/Cycle y al Repository causó una gran expectación e impulsó las investigaciones en esta área; sin embargo, pasado un lustro, esta filosofía y sus productos han resultado un fracaso, siendo su impacto menor que el esperado. A mediados de 1992 este proyecto fue cancelado.

Por otro lado, los problemas de interconexión entre herramientas CASE llevaron a la creación del comité técnico CDIF (CASE Data Interchange Format) dentro de la EIA (*Electronic Industries Association*), que desarrolla una forma de intercambio de datos entre herramientas CASE.

Otros proyectos a destacar en esta área son PCTE y PCTE+ (*Portable Common Tool Environment*), patrocinados por la ECMA (Asociación Europea de Fabricantes de Computadores), que pretende sentar las bases de Entornos Integrados para Soporte de Proyectos (IPSE). Hace unos años han aparecido en el mercado algunos productos basados en el estándar PCTE.

Existen ciertas similitudes entre los estándares PCTE e IRDS, si bien sus orígenes son distintos: el IRDS de ISO proviene de la comunidad de bases de datos, mientras que PCTE tiene su origen en la ingeniería de software. Como señala SAGOLS (1993), "*el PCTE e IRDS de la misma manera que el Fortran y Cobol son específicos del dominio. El primero se refiere a interfaces repositorio/herramienta/aplicación, de la misma manera que el segundo se refiere a lenguajes de diccionario de datos/acceso*".

Desde el punto de vista técnico IRDS ha considerado los datos centralizados y es más adecuado para administración y gestión de datos, mientras que PCTE considera los datos distribuidos, y resulta mejor para definir un entorno de ingeniería de software y para la integración de herramientas.

Debido a estas razones, a continuación profundizamos en los estándares ISO/IEC, remitiendo al lector interesado en los otros estándares a GARBAJOSA y LÓPEZ (1995).

ESTÁNDARES ISO/IEC PARA SDRI

Existen varios estándares promulgados por ISO/IEC para los SDRI, a continuación presentaremos los más importantes.

1. Marco de referencia de SDRI

El objetivo de este estándar, aprobado como ISO/IEC 10027, ISO (1990), es *especificar una herramienta software que pueda ser utilizada para describir y controlar recursos de información de la empresa*, para lo que analiza los datos, los principales procesadores, las interfaces asociadas y los servicios que deben estar contenidos en el RI.

Además de presentar un conjunto de definiciones relativas a los SDRI, y las convenciones utilizadas en los estándares SDRI, se propone una arquitectura recursiva (véase ANSI (1986) —, con cuatro niveles de datos y tres pares de niveles asociados.

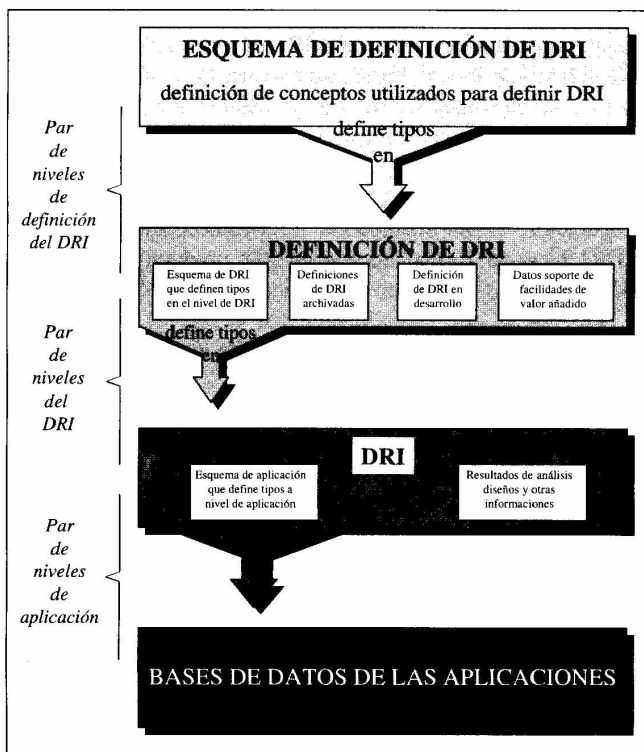


Figura 13.6. Niveles de datos en el SDRI de ISO

En cada nivel se describe, además de otras posibles informaciones, los tipos de objetos que se pueden almacenar en el nivel inferior.

Estos niveles se muestran en la figura. 13.6 y son el esquema de la definición de DRI, el nivel de definición de DRI y el nivel de aplicación (bases de datos de las aplicaciones).

El primero especifica los conceptos primitivos utilizados para definir un DRI, que serán los elementos básicos de un modelo de datos.

El objetivo del segundo nivel es contener las definiciones de DRI. Puede haber varias definiciones de DRI, cada una descrita por un esquema diferente.

Una parte de la definición de DRI, denominada *Esquema de DRI*, prescribe los tipos de objetos acerca de los que se almacenan datos en uno o más DRI. Una definición de DRI puede contener uno o más esquemas de DRI, como puede ser: el esquema de DRI activo, definiciones de diccionarios archivados y de otros en fase de desarrollo; además, puede soportar facilidades de valor añadido (por ejemplo, para el control de versiones).

El objetivo del tercer nivel es contener DRI descritos en el nivel superior. Parte del contenido de DRI define los tipos del nivel de aplicación, y otra información de un DRI incluye esquemas de aplicaciones no activas y resultados de otros análisis, diseños e información de control acerca de las aplicaciones.

El propósito de los datos de un DRI es permitir a un SDRI soportar el diseño, construcción y operación de un sistema de información automatizado. Los datos se cargarán en el DRI bien de forma automática por productos software (por ejemplo, herramientas CASE) o por los diseñadores de aplicaciones.

El DRI contiene también información que permite al administrador de DRI llevar a cabo sus funciones.

Por último, en el nivel de aplicación se almacenan las ocurrencias de los datos que se describen en el DRI, es decir, las bases de datos de la empresa (es el nivel que está orientado al usuario del SI).

Entre estos cuatro niveles se definen tres pares de niveles sobre los que operan los servicios:

- par de niveles de definición de DRI
- par de niveles de DRI
- par de niveles de aplicación

El SDRI proporciona servicios a nivel de definición del DRI y a nivel del DRI; sólo los dos primeros pares son objeto de estandarización por parte del proyecto de SDRI. La

estandarización del último par es responsabilidad del proyecto de lenguajes de bases de datos, poniéndose así de manifiesto la necesaria interrelación entre distintos estándares del grupo WG3, e incluso, entre estándares de distintos grupos.

En este estándar también se describen las facilidades que debe proporcionar un SDRI, entre las que destacan fundamentalmente dos. Por un lado, facilidades generales de gestión de BD, como pueden ser: definición y acceso a los datos, control de integridad, valores límites y por defecto, auditoría, generación de informes, acceso remoto a los datos, etc. y, por otro, facilidades específicas de la gestión de recursos de información, como pueden ser las de denominación, gestión del ciclo de vida, control de versiones, análisis de impacto, etc.

Otro aspecto importante que se trata en el estándar son las interfaces del SDRI, representadas en la figura 13.7. En la figura representamos las interfaces según se proponen en la nueva propuesta UK (1997). Los objetos de la parte de la derecha no están sujetos a normativa por parte de este proyecto.

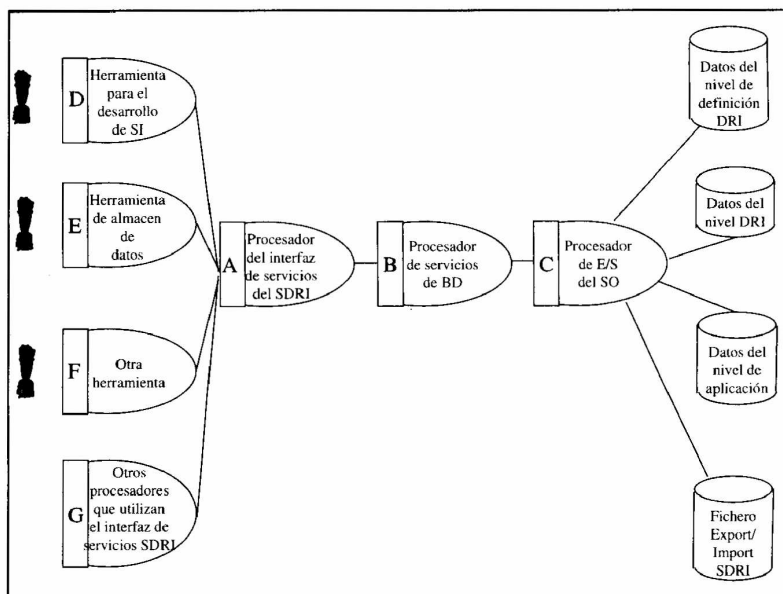


Figura 13.7. Interfaces del SDRI según ISO

Gráficamente la notación que utilizaremos para representar un procesador y la interfaz que ofrece, es la empleada en el estándar: el procesador se representa en forma de *media luna*, en la que se especifica el nombre del mismo. La interfaz se representa mediante un rectángulo con un número o letra que lo identifica.

Como puede verse en la figura, existe una interfaz del procesador de E/S del SO ("C"), que ofrece los servicios habituales de gestión de memoria principal, gestión de

conurrencia, etc. Además, existe la interfaz de servicios de la BD ("B") que ofrece todos los servicios de un SGBD: control de seguridad, integridad, definición y manipulación de datos, etc.

La interfaz fundamental para los SDRI es la denominada interfaz de servicios de SDRI ("A"), que proporciona todos los servicios que necesita un procesador que desee manipular datos del DRI; siendo el medio de acceso a los niveles de DRI y de definición de DRI.

Existen además un amplio abanico de herramientas de desarrollo de SI, herramientas CASE, herramientas de almacén de datos (*datawarehousing*) y otras que comparten el diccionario proporcionado por el SDRI. Estas herramientas pueden ofrecer interfaces hombre-máquina como D, E, y F en la figura, o de otro tipo como G.

7.2. Interfaz de servicios

La interfaz de servicios es, como hemos visto en el epígrafe anterior, uno de los elementos más importantes del entorno del SDRI. El estándar relativo a esta parte ha sido aprobado en 1993, y se conoce como ISO/IEC 10728, ISO (1993), y también se está revisando para incorporar componentes del paradigma de la orientación a objetos.

En él se especifican las estructuras de datos necesarias y los servicios ofrecidos por la interfaz de servicios "A". Las primeras se describen utilizando el modelo relacional (más concretamente el modelo propuesto en el estándar SQL-92, con algunas extensiones, como la generalización entre tablas, esto es, la posibilidad de definir sub/supertablas); mientras que para cada servicio se especifica su función, su formato (en lenguaje Pascal), la entrada, la salida (incluyendo los mensajes de error) y las reglas generales.

Además en este estándar se presenta el mecanismo de gestión de versiones y de configuraciones, basado en el concepto de *conjunto de trabajo* (*working set*), que se define como *"una colección de versiones de objetos de definición u objetos DRI, definidos por un usuario SDRI, como una unidad con fines de gestión de cambios, especificación del estado y control de acceso"*.

7.3. Otros estándares de ISO/IEC

La vinculación de programas escritos en lenguaje C con el SDRI, es objeto de otro estándar que ha empezado a desarrollarse a finales de 1992 y que se ha aprobado como norma internacional en julio de 1995. También, el Departamento de Defensa (DoD) de EEUU, a través de ANSI, ha sometido a estudio por el grupo de expertos de ISO, otro estándar, análogo al que se está desarrollando para C, cuya finalidad es la vinculación de Ada al SDRI y también se ha trabajado en vinculaciones con Java.

Como hemos señalado, el afianzamiento y la creciente penetración de las técnicas de la orientación al objeto, junto con las ventajas que aporta su aplicación, han llevado a pensar en la conveniencia de revisar los modelos de datos en los que se apoyan los estándares de SDRI, yendo hacia modelos de objetos que integren el comportamiento junto con los datos y que proporcionen todas las facilidades de modelado de este paradigma, HODGES (1993).

Otros proyectos sobre los que se está trabajando dentro de la familia de los SDRI de ISO son:

- Guías para el Diseño de Módulos de Contenido SDRI. Se entiende por módulo de contenido *"la especificación para un determinado universo del discurso, de un conjunto de tipos de objetos así como las reglas para gobernar el comportamiento de los mismos"*. Ya existe una propuesta de módulo de contenido para soportar aplicaciones SQL y otro para soportar finalidades de denominación y tesoro.
- Intercambio de datos (exportación/importación) entre SDRI, para el cual se ha propuesto una especialización del estándar ISO/IEC 13238, ISO (1996).

HERRAMIENTAS CASE Y DISEÑO DE BASES DE DATOS

El desarrollo del software ha sido abordado durante años sin el apoyo de unos principios de ingeniería y de unas metodologías, mediante los cuales se consiguiera una mayor productividad y calidad de los productos desarrollados. En los años setenta, se advierten señales de preocupación por estos problemas y aparecen las primeras metodologías y técnicas destinadas a ofrecer soporte al desarrollo de sistemas de información, las cuales empiezan, unos años más tarde a ser integradas en herramientas destinadas a automatizar algunas fases del ciclo de vida.

En este capítulo se estudian las principales características de las herramientas CASE de ingeniería del software asistida por computador, en especial, aquellas más enfocadas al diseño de datos; se realiza una clasificación de las mismas y se resumen algunos prototipos y herramientas para la enseñanza y diseño avanzado de bases de datos.

1. INTRODUCCIÓN

La tecnología conocida con el nombre de **CASE** (*Computer Aided/Assisted Software/System Engineering*) se puede definir como el conjunto de herramientas y metodologías que soportan un enfoque de ingeniería para las distintas fases del desarrollo de software.

Esta tecnología surge a mediados de los años setenta, cuando empiezan a aparecer las primeras metodologías estructuradas y se inician las investigaciones sobre entornos de desarrollo. A mediados de los años ochenta, se populariza y surgen las primeras

herramientas de documentación y diagramación automática. Es una época en la que explodian el número de seminarios, cursos, revistas, libros y congresos dedicados al tema. También ha desempeñado un papel importante en este desarrollo la aparición de las estaciones de trabajo, que aportan una buena interfaz gráfica asociada a una gran capacidad de proceso, dos de los requisitos básicos para el CASE.

Además en la década pasada surge el concepto de repositorio (véase capítulo anterior) como núcleo de un entorno CASE, así como generadores de programas y aplicaciones que automatizan gran parte de las últimas fases del ciclo de vida. En paralelo también aparecen los gestores de proyectos, algunos de los cuales se integran con herramientas de desarrollo.

Sin embargo, esta *primera generación* de herramientas fracasa, debido principalmente a tres factores:

- Limitaciones de los productos.
- Falsas expectativas sobre sus posibilidades.
- Incorrecta implantación.

Actualmente, hemos llegado a una fase de madurez en la que se empieza a implantar una *segunda generación* de herramientas (algunas de las cuales ya no aparecen bajo el término CASE, para no rememorar el fracaso anterior), que superan gran parte de las limitaciones anteriores. Además, los usuarios conocen mejor sus posibilidades y han aprendido a poner unas expectativas más justas sobre éstas, mejorando también los procesos de adopción de metodologías y herramientas.

La tecnología CASE supone la "*informatización de la informática*", es decir "*la automatización del desarrollo del software*", contribuyendo así a elevar la productividad y la calidad en el desarrollo de sistemas de información, de forma análoga a lo que suponen las técnicas CAD/CAM¹ en el área de fabricación.

Este nuevo enfoque a la hora de construir software, persigue los siguientes objetivos:

- Permitir la aplicación práctica de metodologías estructuradas en un primer momento y, en la actualidad, también orientadas a objetos, lo que resulta muy difícil sin emplear herramientas.
- Mejorar la calidad del software.
- Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.
- Simplificar el mantenimiento de los programas.
- Estandarizar la documentación.
- Aumentar la portabilidad de las aplicaciones.

¹ CAD (*Computer Aided Design*), diseño asistido por computador. CAM (*Computer Aided Manufacturing*), fabricación asistida por computador.

- Facilitar la reutilización de componentes software.
- Permitir un desarrollo *visual* de las aplicaciones, mediante la utilización de gráficos.

De una manera muy esquemática, véase figura. 14.1, se puede afirmar que una herramienta CASE se compone de los siguientes elementos:

- *Interfaz de usuario*, que constará de editores de texto y herramientas de diseño gráfico, que permitan mediante la utilización de un sistema de ventanas, iconos y menús, con la ayuda del ratón², definir los diagramas, matrices, etc. que incluyen las distintas metodologías.
- Sistema de gestión del *repositorio* (diccionario) donde se almacenan los elementos definidos o creados por la herramienta, y que se basa en un SGBD o en un sistema de gestión de archivos.
- *Metamodelo* (no siempre visible), que define las técnicas y metodologías soportadas por la herramienta, y que es conveniente que pueda ser extensible por parte del usuario.
- *Generador de informes*, que permite obtener toda la documentación asociada a las técnicas y metodologías.
- Herramienta de *carga/descarga* de datos, que permite cargar el repositorio de la herramienta CASE con datos provenientes de otros sistemas, o bien generar a partir de la propia herramienta esquemas de bases de datos, programas, etc.

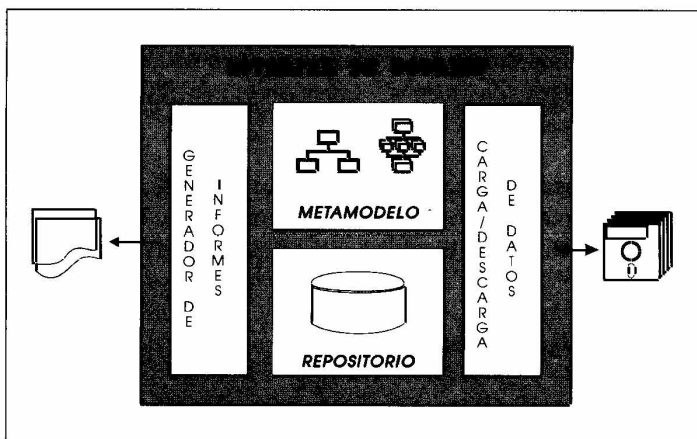


Figura 14.1. Componentes de una herramienta CASE

²Lo que se conoce usualmente por las siglas inglesas WIMP (Windows, Icons, Mouse y Pull-down menus).

2. CATEGORÍAS DE HERRAMIENTAS CASE

Como sucede en otras áreas de la informática, la tecnología CASE emplea una terminología que puede resultar a veces confusa.

En primer lugar se suele distinguir, atendiendo a la fase del ciclo de vida que soportan, entre:

- **CASE frontales** ("*front-end*") o **superiores** ("*Upper CASE*"), abarcan las primeras fases de análisis y diseño.
- **CASE dorsales** ("*back-end*") o **inferiores** ("*Lower CASE*"), cuyo objetivo suele ser el diseño detallado y la generación de código.

Se denomina **ICASE** (*Integrated CASE*) a las herramientas que engloban ambos aspectos, e **IPSE** (*Integrated Programming Support Environment*) a aquellas que, además, incluyen componentes para la gestión de proyectos y la gestión de configuración.

A continuación presentamos de manera muy resumida las categorías de herramientas CASE más frecuentes.

A) Herramientas de análisis y diseño

Dentro de las herramientas de análisis y diseño, destacan las herramientas que permiten crear y modificar diagramas E/R, diagramas de flujo de datos, diagramas de estructura de cuadros, diagramas de clases, etc.

También son muy importantes las herramientas de prototipado como los diseñadores de pantallas, generadores de menús, generadores de informes y lenguajes de especificación ejecutables.

Un aspecto a destacar es la capacidad de análisis y verificación de especificaciones que soporta la herramienta, no sólo sintáctica sino también semántica, como, por ejemplo, la capacidad de normalizar un diagrama de datos (usualmente hasta tercera forma normal).

B) Generación de código y documentación

A partir de las especificaciones del diseño se puede generar código tanto para los programas (por ejemplo, en C, COBOL, C++ o JAVA) como el relativo a los esquemas de bases de datos (sentencias de definición en SQL). Actualmente, las herramientas CASE ofrecen interfaces con diversos lenguajes de cuarta generación para la construcción de sistemas de manera rápida.

Las herramientas CASE también soportan la creación automatizada de un conjunto muy variado de documentación (obtenido a partir de la información almacenada en el rectorio) que va desde la descripción textual de un pseudocódigo hasta diagramas más o menos complejos.

C) Herramientas de prueba

Las herramientas de prueba se conocen también por las siglas CAST (*Computer Aided Software Testing*), y es un área bastante reciente dentro de la tecnología CASE.

D) Herramientas de gestión de configuración

En entornos de desarrollo complejos, especialmente si se integran diversas herramientas de ingeniería de software, se hace imprescindible la incorporación de una herramienta capaz de gestionar la configuración de los sistemas, las distintas versiones de los componentes, etc.

E) Herramientas de ingeniería inversa

Dentro de este apartado destacan diversas herramientas, como las que llevan a cabo:

- **Ingeniería inversa de datos**, que son capaces de extraer la información del código fuente que describe la estructura de los elementos de datos; construyendo así diagramas OO o E/R partiendo de esquemas relacionales, en red o, incluso, archivos.
- **Ingeniería inversa de procesos**, que incluso permiten aislar la lógica de las entidades, y las reglas del negocio a partir del código.
- **Reestructuración de código fuente**, que modifican su formato o implantan un formato estándar.
- **Redocumentación**, que permiten generar diagramas a fin de que se comprenda mejor el código.
- **Análisis de código**, cuyas funcionalidades van desde la identificación automática del código fuente hasta la posibilidad de ir visualizando dinámicamente las llamadas del mismo.

1. HERRAMIENTAS DE DISEÑO DE BASES DE DATOS

1.1. Clases de herramientas

Se pueden utilizar tres clases de herramientas para el diseño de bases de datos:

- **Sistemas o prototipos de investigación**, que resultan, sin duda, los más completos a la hora de soportar las metodologías de diseño y de ofrecer ayudas al diseñador; pero que, como contrapartida, suelen ser difíciles de encontrar y no cuentan con soporte, documentación y evolución adecuados.
- **Herramientas comerciales específicas** para el diseño de bases de datos, dentro de las que destacan muchas desarrolladas por los propios fabricantes de SGBD. Este tipo de herramienta es muy adecuado para aquellos que desarrollen en entornos en los que el componente de datos tenga un peso fundamental, como sucede en muchas aplicaciones de gestión.
- **Herramientas CASE generales**, que incluyen junto con técnicas para el desarrollo de procesos (diagramas de flujo de datos, diagramas de estructuras de cuadros, etc.) otras para el diseño de bases de datos (como el modelo E/R). Constituyen entornos muy completos, integrándose con numerosas herramientas, lenguajes de cuarta generación, generadores de código, etc.

El primer tipo de herramientas suele soportar de forma exhaustiva la fase de modelado conceptual, ofreciendo la posibilidad de crear esquemas a partir del lenguaje natural; así como de *integrar vistas* (técnica que consiste en elaborar esquemas conceptuales parciales y a partir de éstos ir obteniendo esquemas conceptuales mayores que los engloben). Desafortunadamente, existen muy pocas herramientas comerciales que soporten de manera satisfactoria esta técnica.

Las herramientas comerciales específicas para diseño de bases de datos, si bien permiten construir un esquema conceptual bastante completo, se suelen limitar a realizar la detección de sinónimos y homónimos entre las diferentes *vistas*, pero no asisten al diseñador durante el proceso de integración. Donde destacan las herramientas construidas por los fabricantes de SGBD es en los aspectos de diseño lógico específico y de diseño físico de la base de datos que soportan.

Por último, las herramientas CASE generales normalmente ponen más énfasis en el soporte del análisis y construcción de procesos y programas que en la base de datos; a pesar de esto, suelen soportar el diagrama E/R y la producción de esquemas SQL.

3.2. Deficiencias de la tecnología CASE para el diseño de bases de datos

Aunque desde hace un par de años estamos asistiendo a la difusión de una nueva *generación* de herramientas CASE, todavía presentan, tanto las herramientas comerciales como algunos prototipos de investigación, bastante deficiencias en lo que se refiere al diseño de bases de datos; entre los que destacan, REINER (1991):

- Falta de modelado de restricciones, que se suele limitar a soportar las cardinalidades y poco más; mientras que los diseñadores de bases de datos se enfrentan cada día a diseños más complejos. Hay que tener en cuenta que las *reglas de negocio* que se deben modelar son muy variadas y abarcan no sólo las restricciones estáticas sino también las dinámicas, DÍAZ *et al.* (1996). A este respecto hay que destacar que algunas herramientas ofrecen la posibilidad de definir disparadores (*triggers*) asociados a las entidades; sin embargo, en nuestra opinión, esto no resulta del todo adecuado.
- Falta de modelado de rendimiento, ya que no ofrecen medidas o estimaciones de rendimiento. Normalmente tampoco se soporta de forma satisfactoria el diseño físico, ya que se limitan a generar unos cuantos índices, sin tener en cuenta otras posibilidades como el formato de los archivos, replicación, información derivada, etc., GRAEFE (1993).
- Gráficos sin semántica, ya que muchas herramientas *bombardean* al diseñador con multitud de gráficos distintos, soportando múltiples notaciones, pero todo se queda en los diagramas, sin que la herramienta llegue a *comprender* el contenido de los gráficos.

Como se señala en GOTTHARD *et al.* (1992), "*todos estos entornos sufren de las mismas debilidades que los entornos de diseño de otras áreas, enfatizan la mecanización del trabajo rutinario y tedioso como el dibujo..., pero dan poco soporte a los procesos creativos y de toma de decisiones*".

Otra asignatura pendiente de las herramientas es el soporte de los *nuevos* SGBD, que ofrecen la posibilidad de definir y gestionar bases de datos distribuidas, paralelas, activas, orientadas a objetos, etc.; aunque en alguno de estos casos ni siquiera existen metodologías del todo adecuadas para su diseño.

3.3. Resumen de algunos proyectos y herramientas de desarrollo de bases de datos

A continuación presentaremos de forma muy resumida algunos de los proyectos de investigación más importantes en el campo de las herramientas de desarrollo de bases de datos:

OO-Method/CASE, PASTOR, O. *et al.* (1997)

Se trata de una metodología de desarrollo de software orientado a objetos, sustentada por una herramienta CASE, creada por el equipo de Modelado Conceptual Orientado a Objetos y Bases de Datos de la Universidad Politécnica de Valencia. OO-Method cubre las fases de análisis, diseño e implementación de un sistema de información y fusiona el

uso de un enfoque formal para especificación de modelos conceptuales (OASIS) con una notación gráfica estándar utilizada por la mayoría de las metodologías convencionales OO.

CAESAR, DÍAZ *et al.* (1997)

En la Universidad del País Vasco se ha desarrollado un entorno que permite modelar restricciones sobre el comportamiento de bases de datos, con especial énfasis en bases de datos activas. Este entorno soporta la descripción gráfica de *políticas* de negocio, verificando las reglas dinámicas que se definen.

▪ **DAIDA, JARKE *et al.* (1992)**

Este sistema, que forma parte de un proyecto ESPRIT, permite representar conocimiento sobre los requisitos, diseño e implementación de la base de datos, los programas de aplicación y las interfaces. Se compone de seis prototipos: SMLS (formulación de requisitos y prototipado), GraFic (editor gráfico), IRIS (transformación de los requisitos a diseño), PROBE (prototipado rápido basado en diseños TAXIS-DL), DBPL-MAP (refinamiento de especificaciones TAXIS-DL en programas de bases de datos) y ConceptBase (herramientas de integración, control de trabajo, etc.).

DATAID, ALBANO *et al.* (1985)

Es uno de los primeros y más conocidos sistemas de diseño de BD, se desarrolló en Italia conjuntamente por las Universidades de Roma y Milán. DATAID presenta una metodología para el diseño de BD (DATAID-1), parte de la cual está soportada por un conjunto de herramientas con el fin de introducir la filosofía de ayuda por computador dentro del ámbito del diseño de bases de datos. Algunos de los módulos de DATAID son: NLDA (utilizado en análisis de requisitos a partir de lenguaje natural), INCOD-DTE (para la fase de diseño conceptual) y DIALOGO (como entorno de trabajo interactivo que unifica los módulos entre sí).

DDEW, ROSENTHAL y REINER (1994)

Se desarrolló en Computer Corporation of America y consiste en un entorno gráfico que soporta todas las fases del diseño de BD, integrando varias herramientas. Utiliza un riguroso enfoque de transformación de esquemas combinado con heurísticas, intuiciones e interacción con el usuario. Soporta varios modelos de datos, implementados sobre un modelo subyacente unificado que reduce las inconsistencias y redundancias. Contiene, además, un conjunto de algoritmos para reorganizar un esquema sin cambiar su contenido de información. Su principal aportación es ilustrar cómo la teoría puede adaptarse a un sistema práctico.

EDDS, CHOOBINEH et al. (1992)

Este sistema permite construir diagramas E/R a partir del análisis de impresos, descritos a través del FDS (Form Definition System).

LODM, STEINBERG y LIN (1996)

En este sistema sus autores eligen el lenguaje natural como único medio para modelar datos (sin emplear por tanto el modelo E/R ni ningún otro modelo semántico), produciendo esquemas relacionales. Consta de un editor de textos, un *parser*, una herramienta de validación de esquemas y facilidades para exportar esquemas.

LOLITA, MICH y GARIGLIANO (1994)

LOLITA es un sistema de procesamiento de lenguaje natural a gran escala basado en una red semántica que puede analizar texto morfológica, sintáctica, semántica y pragmáticamente, y que se utiliza para generar esquemas orientados al objeto.

MIMOCASE, MARCOS y DE MIGUEL (1997)

Es un entorno para el diseño de BDOO realizado por las Universidades Carlos III de Madrid y la Universidad Rey Juan Carlos. Soporta el modelado conceptual OO y la transformación de esquemas conceptuales a ODMC y CTT utilizando la metodología propuesta en BERTINO y MARCOS (2000).

MUVIS, HAYNE y RAM (1990)

Este sistema está especialmente concebido para soportar la integración de vistas, utilizando una extensión del modelo E/R. Los diseñadores crean las vistas en el VMS (sistema de modelado de vistas) que también se encarga de verificar su integridad; y posteriormente se envían los objetos definidos en la vista a un entorno de integración (VIS). Ofrece la posibilidad de tratar electrónicamente la resolución de conflictos.

OICSI, ROLLAND y PROIX (1992)

Es un sistema experto que genera un esquema conceptual a partir de una descripción en un subconjunto del francés. Partiendo de una interpretación del lenguaje natural produce una primera versión de una red semántica correspondiente a un esquema conceptual intermedio. OICSI aplica su conocimiento sobre diseño de bases de datos a la red semántica para buscar inconsistencias, homónimos y falta de compleción. Su base de reglas contiene cinco tipos de reglas: de análisis del lenguaje natural, de interpretación, estructurales, de validación y de diálogo.

RIDL*, DE TROYER (1989)

Desarrollado en la universidad de Tilburg, se compone de tres módulos: RIDL-G, que soporta el diseño conceptual en el modelo NIAM; RIDL-A, que permite la validación de esquemas, y RIDL-M, que genera esquemas relacionales.

SECSI, BOUZEGHOUB y GARDARIN (1984)

Este sistema se concentra en el diseño del esquema lógico con restricciones de integridad. Como característica importante destaca un módulo de diálogo que permite al usuario describir la aplicación utilizando un lenguaje declarativo formal, un subconjunto del inglés y una interfaz gráfica. Este módulo también permite al usuario interactuar con el sistema para detectar y resolver inconsistencias (por ejemplo, sinónimos y homónimos). Los otros módulos que forman el sistema son una base de reglas, una base de hechos, una base de restricciones y un mecanismo de inferencia.

VCS, STOREY y GOLDSTEIN (1988)

Desarrollado en la Universidad British Columbia resulta muy interesante desde el punto de vista de la interacción con el usuario, no tanto por la calidad de la interfaz (textual), sino más bien por la forma en que el sistema extrae información del usuario y razona con ella. Propone una metodología que formaliza la tarea de especificar vistas (esquemas relacionales) y que emplea ideas del Modelo E/R, de abstracción de datos, SMITH y SMITH (1977), así como de la metodología de Especificación Interactiva, BALDISSERA *et al.* (1979). VCS se basa en un diálogo con el usuario para la identificación de entidades, interrelaciones y atributos. Contiene además tutoriales para explicar conceptos en los puntos apropiados de una sesión y realiza muchas comprobaciones sobre el modelo, incluso utilizando técnicas PLN.

WISE, HALPIN (1982)

Es un entorno desarrollado en la Universidad de Queensland (Australia) como sistema de ayuda a la creación de esquemas conceptuales y posterior diseño lógico. Está centrado sólo en datos y consta de: un editor de esquemas conceptuales que opera en modo textual o gráfico; un generador de esquemas conceptuales a partir de oraciones en lenguaje natural; un comprobador de calidad para normalizar por síntesis y validar las restricciones; un optimizador de esquemas conceptuales; y de un traductor de esquemas conceptuales a esquemas relacionales u orientados al objeto

Existen otros sistemas que, en general, son herramientas que resuelven alguna tarea puntual: **E-R Translator**, BRIAND *et al.* (1985), traduce un esquema E/R a un diseño lógico utilizando una red semántica; **AVIS**, WAGNER (1989), y **CARS**, DEMO y TILLI (1986), realizan integración de vistas; **CAERM**, HAWRYSZKIEWYCZ (1985), valida

esquemas E/R mediante un diálogo con el usuario en lenguaje natural y **FD-EXPERT**, RAM (1995), obtiene dependencias funcionales a partir de un esquema E/R.

4. MARCO PARA LA EVALUACIÓN DE HERRAMIENTAS DE DISEÑO DE BASES DE DATOS

Dada la gran heterogeneidad de herramientas existentes para el diseño de bases de datos, han aparecido varias propuestas para su clasificación (véase, por ejemplo, STOREY y GOLDSTEIN (1993), CHEN *et al.* (1982) o NAVATHE (1985)). A continuación presentamos el marco propuesto en RAM (1994) por considerar que es uno de los más completos.

Ram propone evaluar las herramientas de diseño de bases de datos según 18 parámetros:

- *Origen*: que permite distinguir entre herramientas desarrolladas en entornos académicos de las creadas en entornos comerciales.
- *Fases de desarrollo soportadas*.
- *Modelo/conceptos subyacentes*, que pueden ser semánticos, orientados a objetos, basados en formas, lenguaje natural, relacional, etc.
- *Metodología/algoritmos utilizados*, que examina las reglas o los heurísticos empleados para producir un diseño.
- *Entradas*, describe el tipo de entradas que requiere la herramienta.
- *Salidas*, describe las salidas producidas por la herramienta.
- *Medios de representación/Interfaces*, este parámetro identifica interfaces gráficas y textuales de la herramienta.
- *Repositorio de información*, en el sentido de conocer qué diseño y estructura utiliza (archivos, SGBD relacional, SGBD, etc.).
- *Documentación*.
- *Análisis de alternativas*, si la herramienta presenta alternativas y ayuda al diseñador a tomar decisiones.
- *Verificación y validación del diseño*.

- *Público*, por ejemplo, usuarios finales, diseñadores, etc.
- *Validación/Difusión de uso*, para conocer en qué casos se ha utilizado.
- *Características operativas*, identifica el entorno hardware en el que opera, el lenguaje en que ha sido desarrollada, etc.
- *Facilidad de modificación*, examina la capacidad de la herramienta para llevar a cabo cambios en la herramienta.
- *Facilidad de reutilización*.
- *Extensiones futuras*.
- *Comentarios generales*.

5. ENEAS/BD: UN ENTORNO PARA LA ENSEÑANZA AVANZADA DE SISTEMAS DE BASES DE DATOS

En el grupo de investigación LABDA (Laboratorio de Bases de Datos Avanzadas) de la Universidad Carlos III de Madrid, se desarrolló el sistema ENEAS/BD (Entorno para la Enseñanza Avanzada de Sistemas de Bases de Datos), DE MIGUEL *et al.* (1996), cuya arquitectura se muestra en la figura 14.2.

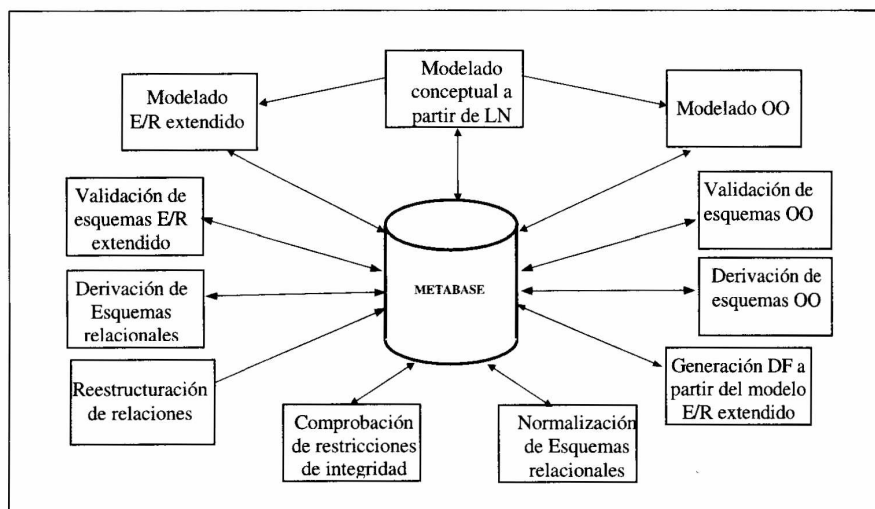


Figura 14.2 Arquitectura de ENEAS/BD

La idea de crear un entorno de desarrollo enfocado a la enseñanza del diseño de BD, surge de la necesidad de disponer de un conjunto de herramientas automatizadas que permitan al usuario adquirir y, posteriormente, afianzar conocimientos en el área de BD a través de todas las fases del proceso de creación de una BD.

ENEAS/BD es una herramienta concebida con dos objetivos principales: ayudar al usuario en el diseño de BD y proporcionar un entorno de aprendizaje en este campo. Creemos que ENEAS/BD introduce mejoras respecto a otros sistemas estudiados, ya que, además de incluir facilidades para el aprendizaje, soluciona parte de las limitaciones encontradas en algunos de los sistemas previamente analizados. No se trata de un conjunto de herramientas aisladas sino de un entorno con una arquitectura modular e integrada, en la cual el diseñador (bien sea experimentado, bien en formación) constituye el eje fundamental hacia donde se enfoca todo el sistema, puesto que hemos considerado de importancia capital una buena interacción con el mismo.

Los objetivos planteados en el desarrollo de ENEAS/BD son:

- **Extensibilidad:** Posibilidad de incorporar fácilmente nuevas funcionalidades sin que se pierda la visión global del sistema.
- **Portabilidad:** Sistema utilizable en distintas plataformas.
- **Integrabilidad:** Todos los módulos se relacionan a través de una metabase que mantiene la información que todos ellos comparten.
- **Independencia:** La integrabilidad no implica que el usuario no pueda utilizar los módulos individualmente.
- **Flexibilidad:** Debido a la independencia y a las facilidades que suministra el sistema, el usuario podrá componer la metodología que mejor se adapte a sus necesidades.
- **Interfaz homogénea y fácil de usar** que facilita el diálogo con el usuario.

Para la caracterización de ENEAS/BD nos basamos en el marco para la evaluación de herramientas de diseño de BD automatizado propuesto en RAM (1994), que como hemos señalado, selecciona 18 parámetros de los que hemos recogido y adaptado, en su caso, aquellos aplicables al estado actual del proyecto ENEAS/BD:

1. **Origen.** Aunque ENEAS/BD es una herramienta desarrollada en un entorno académico, se ha enfocado para que su aplicación no se limite sólo a este entorno, sino que sea también utilizable en el ámbito industrial.

2. **Fases de desarrollo soportadas.** ENEAS/BD cubre las fases de análisis conceptual, diseño lógico y diseño físico.
3. **Modelo subyacente.** ENEAS/BD incorpora un modelo unificado para todas las fases de desarrollo de una base de objetos denominado **Metamodelo para la Integración de Modelos de Objetos (MIMO)**, MARCOS (1997), el cual se utiliza en la representación de la Metabase.
4. **Metodología soportada.** ENEAS/BD proporciona una metodología propia que guía al usuario a lo largo de las fases de diseño. Además le permite la posibilidad de modificar la metodología con el fin de adaptarla a sus necesidades.
5. **Métodos utilizados.** Además de algoritmos clásicos y mejorados (normalización) se incorporan nuevos algoritmos, reglas y heurísticas utilizadas para validación, refinamiento, etc. Por otro lado, se utilizan técnicas de procesamiento de lenguaje natural como método de extracción de información.
6. **Entradas/Salidas.** Este parámetro tiene un alto grado de libertad en ENEAS/BD, puesto que ofrece la posibilidad de obtener la misma salida partiendo de distintas entradas (por ejemplo, el modelo E/R puede obtenerse a partir de especificaciones en LN o a partir de dependencias funcionales). Igualmente, a partir de una misma entrada podrán obtenerse distintas salidas (por ejemplo, a partir de una especificación en LN podrá obtenerse un esquema E/R o un esquema OO).
7. **Principales representaciones e interfaces.** En ENEAS/BD, al tratarse también de un entorno de enseñanza, se da mucha importancia a su mecanismo de interacción con el usuario. Se ha definido un estándar de interfaz gráfica de usuario (IGU) utilizada por todos los módulos con el fin de ofrecer un modo uniforme de comunicación con el sistema, independientemente del módulo al que se esté accediendo. Esta interfaz permite la comunicación a través de un entorno de ventanas, ofreciendo diversos sistemas de ayuda (interactiva, menús, etc.) entre los que destaca un *experto* que guía al usuario en el proceso de diseño a la vez que le introduce en la herramienta.
8. **Repositorio de información.** Este parámetro se refiere a la implementación de la Metabase en la que se almacena toda la información procedente de los distintos módulos así como sus metaestructuras.
9. **Documentación.** Analiza la documentación generada para cada diseño. ENEAS/BD proporciona tanto documentación textual como gráfica, en soporte visual o impreso, dependiendo del módulo y de las preferencias del usuario.

10. **Posibilidad de alternativas** . Evalúa la generación de alternativas al diseño por parte de la herramienta. ENEAS/BD ofrecerá la posibilidad de sugerencias de diseños mejorados mediante un diálogo inteligente que se establecerá con el usuario.
11. **Validación de esquemas**. ENEAS/BD incorpora comprobación, mediante heurísticas y algoritmos, de esquemas E/R y esquemas OO.
12. **Audiencia**. La herramienta está dirigida tanto a usuarios sin ningún tipo de experiencia en el diseño de BD (recuérdese que es también una herramienta orientada al aprendizaje) como a usuarios experimentados, quienes también se podrán beneficiar de las facilidades proporcionadas por ENEAS/BD.
13. **Características operacionales**. ENEAS/BD se ha diseñado para trabajar sobre PC (equipado con procesador i486 o superior) en entorno Windows. La implementación del modulo de PLN se ha realizado en PROLOG y LISP, el resto de los módulos se implementan en C++, mientras que la Interfaz Gráfica de Usuario y la integración de todos los componentes se realiza en Delphi. Su diseño se está realizando de forma que en un futuro sea fácilmente portable a otras plataformas.
14. **Facilidades de cambio y mantenibilidad**. Determina las posibles repercusiones que la modificación de la información manejada en un determinado módulo tiene en el entorno. Debido a la existencia de la Metabase, la información está permanentemente actualizada a disposición de todos los módulos.
15. **Reutilización**. Se mantendrá un repositorio con diseños obtenidos de experiencias anteriores que será utilizado para ayudar al usuario en los nuevos diseños.
16. **Validación del sistema**. Con el fin de demostrar y mejorar las capacidades de ENEAS/BD, el entorno se ha validado en distintos ámbitos docentes e industriales.

- A. Ejercicios propuestos**
- B. Ejemplo completo**
- C. Manual de usuario de la
herramienta Reno V.3**
- D. Lista de Acrónimos y
abreviaturas**

Bibliografía

Índice alfabético

**A
P
É
N
D
I
C
E
S**

EJERCICIOS PROPUESTOS

A. EJERCICIOS DE MODELADO

Para cada uno de los siguientes enunciados, se pide :

- **Diseño conceptual** de la base de datos anteriormente descrita utilizando el modelo E/R extendido.
- Realizar el paso del modelo E/R extendido al **modelo Relacional**, obteniendo el grafo relacional.

A.1. La cadena de Videoclubs Glob-Gusters ha decidido, para mejorar su servicio, emplear una base de datos para almacenar la información referente a las películas que ofrece en alquiler. Esta información es la siguiente:

- Una película se caracteriza por su título, nacionalidad, productora y fecha (Por ejemplo, *Quo Vadis*, *Estados Unidos*, *M.G.M.*, 1955).
- En una película pueden participar varios actores (nombre, nacionalidad, sexo), algunos de ellos como actores principales.
- Una película está dirigida por un director (nombre, nacionalidad).
- De cada película se dispone de uno o varios ejemplares diferenciados por un número de ejemplar y caracterizados por su estado de conservación.

- Un ejemplar se puede encontrar alquilado a algún cliente (nombre, dirección, teléfono). Se desea almacenar la fecha de comienzo del alquiler y la de devolución.
- Cada socio puede alquilar como máximo 4 ejemplares.
- Un socio tiene que ser avalado por otro socio que responda de él en caso de tener problemas en el alquiler.

A.2. La asociación "Amigos de la Fiesta" desea recoger en una base de datos toda la información acerca de las corridas de toros que se celebran en España y de todos los datos relacionados con ellas.

- Se desea tener información acerca de cada corrida, identificada conjuntamente por un número de orden, la feria en la que se celebra y el año de celebración (por ejemplo: orden = 2, feria = San Isidro, año = 1990).
- En una determinada corrida actúan una serie de toreros (mínimo 1 y máximo 3) de los que se desea guardar su DNI, nombre, apodo y fecha en que tomó la alternativa (fecha en la que se convirtió en matador de toros). Además se desea saber quién fue el torero que le dio la alternativa (padrino) en su día (un torero puede dar la alternativa a varios compañeros o a ninguno).
- En cada corrida un torero obtiene una serie de premios (cuántas orejas, cuántos rabos y si salió por la puerta grande o no) de los que se desea mantener información.
- Cada torero puede tener un apoderado del que es protegido. A su vez, un apoderado lo puede ser de varios toreros. De él se desea saber su DNI, nombre, dirección y teléfono.
- Una corrida se celebra en una plaza de toros de la que se desea saber su nombre que se supone único, localidad, dirección y aforo. En una misma plaza se pueden celebrar varias corridas de toros.
- En cada corrida son estoqueados al menos 6 toros. Cada toro viene identificado por el código de la ganadería a la que pertenece, el año en que nació y un número de orden. Además se desea mantener información acerca de su nombre y color así como del orden en que fue toreado.
- Cada toro pertenece a una ganadería determinada. De cada ganadería se pretende saber su código, nombre, localidad y antigüedad (fecha de creación).

A.3. El Ministerio de Educación y Ciencia desea mantener información acerca de todos los cuadros que se encuentran en las pinacotecas españolas y toda la información relacionada con ellos.

- De cada pinacoteca se desea saber el nombre (que se supone único), la ciudad en que se encuentra, la dirección y los metros cuadrados que tiene.
- Cada pinacoteca tiene una serie de cuadros de los que se quiere mantener información acerca de su código, nombre, dimensiones, fecha en que fue pintado y técnica utilizada.
- Cada cuadro es pintado por un determinado pintor (nombre, país, ciudad, fecha de nacimiento y fecha de defunción). Un pintor puede tener a otro como maestro; a su vez, un maestro puede serlo de varios (o de ninguno).
- Los pintores pueden pertenecer o no a una escuela de la que se desea saber su nombre y en qué país y en qué fecha apareció.
- Los pintores pueden tener también uno o varios mecenas que les protegen (nombre, fecha, país y ciudad de nacimiento y fecha de muerte). A su vez un mismo mecenas puede serlo de varios pintores. Se desea saber cuál es la relación que existe entre el pintor y su mecenas.

A.4. En el gimnasio “Siempre en forma” se quiere implantar una base de datos para llevar el control de los socios, recursos utilizados, etc. Las especificaciones que nos han dado son las siguientes:

- Existen varias salas, de las cuales se quiere guardar información, como los metros cuadrados que ocupa, ubicación y el tipo de sala (cardio, general, muscular). Cada sala se identifica por un número.
- Hay salas que tienen aparatos y salas que no. En las salas se pueden o no impartir clases.
- Cada aparato está asignado a una única sala, y de cada uno de ellos se quiere tener almacenado su código, descripción y estado de conservación.
- También se quiere mantener información relacionada con las clases que se imparten (descripción y día/hora en la que se imparten); cada clase se identifica por un código de clase. Cada clase tiene asignada una sala en la que se imparte y un monitor.

- De cada monitor se quiere conocer el DNI, nombre, teléfono, si tienen o no titulación y experiencia profesional, así como las clases que pueden impartir (preparación como monitor de aerobio, *step*, *stretching*, etc.).
- De cada socio se quiere conocer el número de socio, nombre, dirección, teléfono, profesión y datos bancarios, así como las clases a las que asiste.
- El gimnasio dispone también de pistas de squash, de las que se quiere conocer el número de pista, ubicación y estado. Las pistas de squash pueden ser utilizadas por socios, y existe un servicio de reserva de pista (en una fecha y a una hora.)

A.5. El gimnasio “siempre en forma” del ejercicio A.4 impone las siguientes restricciones :

- Las pistas de Squash se consideran salas.
- Las clases sólo se imparten en salas sin aparatos. Las salas con aparatos siguen clasificándose en sala de cardio, general, etc.

A.6. El gimnasio “siempre en forma” del ejercicio A.5 elimina la restricción de que una clase sólo se imparten en salas sin aparatos.

A.7. La gestión de una farmacia requiere poder llevar control de los medicamentos existentes, así como de los que se van sirviendo, para lo cual se pretende diseñar un sistema acorde a las siguientes especificaciones:

- En la farmacia se requiere una catalogación de todos los medicamentos existentes, para lo cual se almacenará un código de medicamento, nombre del medicamento, tipo de medicamento (jarabe, comprimido, pomada, etc.), unidades en stock, unidades vendidas y precio. Existen medicamentos de venta libre y otros que sólo pueden dispensarse con receta médica.
- La farmacia compra cada medicamento a un laboratorio, o bien los fabrica ella misma. Se desea conocer el código del laboratorio, nombre, teléfono, dirección y fax, así como el nombre de la persona de contacto.
- Los medicamentos se agrupan en familias, dependiendo del tipo del tipo de enfermedades a las que dicho medicamento se aplica. De este modo, si la farmacia no dispone de un medicamento concreto, puede vender otro similar aunque de distinto laboratorio.
- La farmacia tiene algunos clientes que realizan los pagos de sus pedidos a fin de cada mes (clientes con crédito). La farmacia quiere mantener las unidades

de cada medicamento comprado (con o sin crédito) así como la fecha de compra. Además, es necesario conocer los datos bancarios de los clientes con crédito, así como la fecha de pago de las compras que realizan.

A.8. Se trata de realizar el diseño de la base de datos (en el modelo E/R) para un organismo que desea llevar un control del mantenimiento de sus equipos hardware (computadores, impresoras, etc.), licencias de software (sistemas operativos, SGBD, compiladores, aplicaciones, etc.) y redes.

- Hay que tener en cuenta que en los equipos hardware pueden encontrarse distintas licencias de software, y que se considera que los distintos equipos hardware se pueden encontrar conectados a una red (como máximo) o ser independientes. El software de red se considera un tipo especial de software que se encuentra asociado a la red.
- Las redes se encuentran distribuidas en plantas, debiendo existir un usuario responsable de ellas, lo mismo que para cada uno de los computadores del organismo. También interesa conocer qué aplicaciones utiliza cada uno de los usuarios del organismo. Los usuarios se agrupan en secciones, que a su vez se agrupan en departamentos.
- Para todos los elementos del sistema se almacenará la siguiente información: código, fecha de adquisición, precio de compra y si se tiene o no contrato de mantenimiento. Además, para los computadores se debe almacenar el tipo de procesador, el fabricante y si posee o no disquetera y de qué tipo (téngase en cuenta que algunos computadores pueden disponer de varios tipos: 3'5, 5'25, etc.). Para el software se debe almacenar, en el caso de las aplicaciones, el (o los) lenguaje(s) con las que se han desarrollado, siempre que se conozca.

A.9. Se trata de diseñar la base de datos para la administración de un consorcio de hospitales, que permita gestionar datos acerca del personal así como de sus pacientes. De cada hospital interesa almacenar además su nombre dirección, teléfonos, fax, etc.

- El personal de los hospitales (del que se quiere conocer su DNI, nombre, apellidos, dirección y teléfono) se divide en personal administrativo y personal sanitario (dentro de éste se distingue a su vez entre ATS y médicos).
- Los médicos tienen una especialidad que interesa conocer (pediatría, obstetricia, etc.) y sólo trabajan, al igual que el resto del personal, en un hospital.
- Los pacientes pueden acudir a varios hospitales del consorcio y ser atendidos por varios médicos.

- El médico comunica los datos personales de los pacientes que van a ingresar en el hospital, así como la fecha de admisión y la sala (habitación) en la que deben permanecer. Cada sala se identifica por un número dentro de cada hospital y se desea conocer el número de camas de las que dispone cada una de ellas. No es relevante saber quién es el médico que realiza la admisión.
- El paciente puede modificar estos datos, además de completarlos con otros como el número de seguridad social, compañía aseguradora, etc.
- Cada admisión de un paciente en el hospital lleva asociada una o varias fichas de tratamiento en las que se indica la enfermedad y el médico que la atiende. Cada tratamiento se identifica por el nombre de la enfermedad del tratamiento que es único para cada admisión.
- Además, cada tratamiento da lugar a distintos resultados que permiten realizar el seguimiento de cada enfermedad de un paciente. El resultado debe indicar la fecha y hora en que tuvo lugar, así como un comentario (por ejemplo, indicando si el paciente tiene fiebre etc.). Para un mismo tratamiento sólo puede haber un resultado en un mismo día, a una misma hora.

A.10. Una gran multinacional dedicada al ramo de la venta, alquiler y reparación de vehículos desea diseñar una BD para informatizar parte de su gestión. En una primera fase sólo quiere contemplar los subsistemas de venta y alquiler, dejando el de talleres y reparación para una 2ª fase. Los supuestos semánticos son los siguientes:

- La empresa tiene una serie de concesionarios distribuidos por toda la geografía nacional. Todos los concesionarios se dedican tanto a la venta, como al alquiler y a la reparación de vehículos. Existe un código de concesionario (COD_C).
- Los vehículos, que se identifican por un código (COD_V), pueden pertenecer a un concesionario y estar destinados tanto a la venta como al alquiler, o bien a ambas cosas.
- Los clientes, que se identifican por un código y tienen un nombre, una dirección, etc., pueden comprar o alquilar vehículos a los concesionarios. Sólo interesa conocer los vehículos que un cliente tiene alquilados en el momento actual.
- A pesar de que en esta fase no se va a informatizar el subsistema de reparaciones, sí interesa tener también en la BD, a fines de márketing, los vehículos que sin ser actualmente propiedad de ningún concesionario, sí han tenido algún tipo de relación con alguno de ellos (por ejemplo, porque han estado en alguno de sus talleres o porque han sido vendidos por un concesio-

nario); se desea conocer asimismo los clientes que son propietarios de estos vehículos.

- Los concesionarios pueden cederse vehículos, de forma que alguno de ellos puede pasar de ser propiedad de un concesionario a ser propiedad de otro (interesa guardar el histórico con las fechas en las que un vehículo ha sido propiedad de un concesionario y cuando ha dejado de serlo).
- Los concesionarios pueden tener puntos de venta.
- Existen concesionarios que se encargan de dirigir las operaciones de todos los concesionarios de su zona.
- Los empleados de un concesionario, identificados por un código (COD_E), se dedican a distintas funciones, no pudiendo éstas simultanearse. Estas funciones son tres: venta o alquiler, reparaciones, y por último funciones administrativas. Cada empleado tiene un jefe directo que se encarga de supervisar su trabajo.
- Los empleados pueden comprar vehículos en los concesionarios, pero no pueden alquilarlos por política de la empresa. Por tanto, un empleado puede ser considerado un tipo especial de cliente.

A.11. La empresa “X” desea llevar un control de sus departamentos, empleados y proyectos según las siguientes especificaciones:

- Se desea conocer el nombre, salario y número de la seguridad social de cada empleado, así como el nombre, fecha de nacimiento y estudios que cursa cada uno de sus hijos. Existen tres tipos de empleados: directores (encargados de un departamento), representantes de ventas (se ocupan de la representación en un número de regiones) e ingenieros (encargados de realizar los proyectos de la empresa). Un director no puede ejercer ninguna otra función; sin embargo, un representante de ventas puede desempeñar también las funciones de un ingeniero y viceversa.
- Los distintos departamentos concede becas de estudio a los hijos de los empleados. Se desea conocer la fecha de concesión de cada beca así como la cuantía de ésta.
- Un ingeniero puede tener varias especialidades que se desean conocer.
- De los departamentos se necesita: el nombre, localización y empleados que trabajan en él. Un departamento tiene, como mínimo, 2 empleados, y como

máximo 30, y está al cargo de un único director. Cada departamento tiene un director distinto.

- Un departamento puede controlar un número de proyectos, de los que se desea conocer su nombre y fecha de comienzo.
- En la realización de un proyecto no puede haber involucrados más de 5 ingenieros. Todo ingeniero debe estar asociado a 1 proyecto como mínimo y a 2 como máximo.

A.12. Se trata de diseñar una base de datos para una red de agencias franquiciadas a TECHNOHOUSE, empresa especializada en el alquiler y compra de inmuebles.

- Cada agencia tiene un titular propio y un conjunto de vendedores. Tanto el titular como los vendedores sólo pueden pertenecer a una agencia. Sobre las agencias interesa almacenar su dirección, teléfonos (que pueden ser varios), fax, etc. Además, cada agencia tiene asignada una zona de actuación que es única.
- Las agencias disponen de inmuebles tanto para alquilar como para comprar (incluso ambas cosas), en el primer caso figurará el precio de alquiler y la fianza a depositar, mientras que en el segundo caso, además del precio de venta, se indica si el inmueble está o no hipotecado.
- Por otro lado, los inmuebles pueden ser locales comerciales, o pisos. En ambos casos se identifican por un código, e interesa conocer el propietario, la dirección y la superficie en m².
- Además, en el caso de pisos interesa conocer el número de habitaciones (incluyendo el salón), el número de cuartos de baño, el tipo de gas (natural, ciudad, butano), y si es interior o exterior. Para los locales comerciales se debe conocer si dispone de licencia de apertura.
- Un cliente puede acudir a varias agencias, en cada una se le asigna un vendedor, que es el encargado de seleccionar los inmuebles que cumplen las características deseadas, y en caso de estar interesado, el cliente debe dar una señal para reservar el inmueble (o los inmuebles) que desea.

A.13. Una compañía aseguradora de tipo sanitario desea diseñar una BD para informatizar parte de su gestión hospitalaria. En una primera fase sólo quiere contemplar los siguientes supuestos semánticos:

- Los hospitales de su red pueden ser propios o concertados; además de unos datos comunes a todos ellos como son el código de hospital (CÓD_H), su

nombre (N_H), número de camas (NÚM_C), etc., cuando el hospital es propio se tienen otros específicos como el presupuesto (P), tipo de servicio (TS), etc.

- Una póliza, que se identifica por un número de póliza (CÓD_P), tiene varios atributos que, en principio, no interesa especificar y que se agrupan bajo el nombre de datos de póliza (DATOS_P). Una póliza cubre a varios asegurados, los cuales se identifican por un número correlativo (NÚM) dentro del código de póliza y tienen un nombre (NA), fecha de nacimiento (FN), etc.
- Los asegurados cubiertos por una misma póliza pueden ser de distintas categorías. Mientras los asegurados de primera categoría (A1C) pueden ser hospitalizados en cualquier hospital, los de segunda categoría (A2C) sólo pueden ser hospitalizados en hospitales propios. Aunque las otras categorías no tienen derecho a hospitalización, en la BD se guardan todos los asegurados sea cual sea su categoría.
- Interesa saber en qué hospitales han estado (o están) hospitalizados los asegurados, el médico que prescribió la hospitalización, así como las fechas de inicio (FI) y de fin (FF) de la misma.
- Existen áreas, identificadas por un código (CÓD_A) y con datos sobre su superficie (S), número de habitantes (NÚM_H), etc. Los hospitales concertados tienen que estar asignados a una única área, que no puede cambiar, mientras que los propios no están asignados a áreas.
- Los médicos, que se identifican por un código (CÓD_M), tienen un nombre (N_M), teléfonos de contacto, etc. Interesa conocer las áreas a las que está adscrito un médico. Existe una dependencia jerárquica entre médicos de forma que un médico tiene un único jefe.

A.14. El departamento de formación de una empresa desea construir una base de datos para planificar y gestionar la formación de sus empleados.

- La empresa organiza cursos internos de formación de los que se desea conocer el código de curso, el nombre, una descripción, y el número de horas de duración.
- Un curso puede tener como prerrequisito haber realizado otro(s) previamente, y, a su vez la realización de un curso puede ser prerrequisito de otros. Un curso que es prerrequisito de otro puede serlo, de forma obligatoria u opcional, pudiendo exigirse a veces haberlo realizado en un cierto periodo de tiempo anterior.

- Un mismo curso tiene diferentes ediciones, es decir, se imparte en diferentes lugares, fechas y con diferentes horarios (intensivo, de mañana o de tarde).
- Los cursos se imparten por personal de la propia empresa. Un empleado podrá ser docente si está capacitado para impartir cursos, y/o alumno, si tiene el nivel suficiente para recibirlos, pudiendo existir empleados que no estén capacitados para ninguna de las dos tareas.
- De los empleados que son docentes se desea guardar la información relativa a su historial en el que conste cada una de las materias sobre las que puede impartir cursos, el nivel de experiencia en dicha materia y otras características que puedan ayudar a conocer la capacidad del profesor para impartir esa materia.
- De los empleados que pueden recibir cursos se desea guardar el currículum que los capacita para ello (estudios realizados, años de antigüedad en la empresa, etc.).
- De todos los empleados se desea conocer su código de empleado, nombre y apellidos, dirección, teléfono y Número de Identificación Fiscal (N.I.F.).
- Los cursos que tienen una parte práctica son impartidos por varios profesores. Nos interesa guardar información de cuándo un docente participa como profesor de teoría y cuándo lo hace como profesor de prácticas.
- Un mismo empleado puede ser docente en una edición y alumno en otra, pero nunca puede ser ambas cosas a la vez (en una misma edición de curso o lo imparte o lo recibe).
- Los cursos pueden utilizar ciertos recursos para poder ser impartidos (transparencias, pantalla de cristal líquido, etc.). Se desea guardar la información relativa a éstos (nombre del recurso, descripción, ubicación de éste, unidades de las que se dispone), así como conocer el número de unidades que se desean, no pudiendo éste sobrepasar el número de unidades disponibles. Además, interesa saber si el recurso es necesario, o sólo útil, para que el curso puede llevarse a cabo.

B. EJERCICIOS DE NORMALIZACIÓN

B.1. La Seguridad Social desea conocer los pacientes (DNI) que han sido atendido en sus hospitales (CÓD_H) y el doctor (CÓD_D) que los atiende.

Se supone que un doctor sólo puede atender en un hospital y que, aunque un paciente puede ser atendido en varios hospitales, en cada uno de ellos sólo le atiende un doctor.

Determinar las dependencias funcionales de este supuesto y la forma normal de la correspondiente relación, así como analizar si un diseño alternativo podría ser más adecuado.

B.2. Se desea diseñar una base de datos en el modelo relacional para una universidad, teniendo los siguientes supuestos:

Un profesor se identifica por un código de profesor (CP) y todos los profesores tienen nombres (NP) distintos. Un profesor puede tener varios títulos (T) e intervenir en distintos proyectos (P), no exigiéndose ningún título determinado para intervenir en un cierto proyecto.

Cada asignatura (A) tiene un único profesor como responsable, si bien un mismo profesor puede ser responsable de más de una asignatura. Las asignaturas se dividen en uno o más grupos (G). Todo alumno (AL), en cada asignatura, pertenece a un único grupo.

Cada profesor depende siempre y únicamente de un departamento (D). Así mismo, toda asignatura está ligada a un único departamento, el del profesor responsable de la misma.

Determinar las dependencias funcionales de este supuesto, y la forma normal de la correspondiente relación, y aplique el proceso de análisis para obtener un esquema relacional en la forma normal que considere más conveniente.

B.3. Un departamento universitario desea diseñar una base de datos para la gestión de los cursos que imparte durante un cuatrimestre. En la base de datos quiere almacenar los profesores (P), los estudiantes (E), la nota (N) con la que se califica a un alumno en cada asignatura (AS), así como los días de la semana/hora (H) en las que se imparte una asignatura y el aula (AU) (se supone que ni el día/hora ni el aula en los que se imparte una asignatura varían de una semana a otra). Se desea almacenar también el teléfono (TL) y el despacho (D) de cada profesor (se supone que no existen teléfonos compartidos por dos profesores y que en cada despacho sólo hay un profesor y un teléfono). Se sabe asimismo que un profesor imparte varios grupos (G) y en todos ellos utiliza los mismos textos (T).

Además de los anteriores se dan los siguientes supuestos semánticos:

- En un momento dado tanto un estudiante como un profesor sólo pueden estar en un aula.

- En un momento dado en un aula sólo se puede impartir una asignatura.
- En cada despacho hay un solo teléfono.
- Un estudiante no puede asistir a las clases de dos asignaturas en una misma hora.
- Todas las asignaturas están divididas en los mismos grupos, utilizándose en todos los grupos de la misma asignatura los mismos textos.

Se pide:

1. Determinar las dependencias funcionales. Si alguna dependencia es redundante o alguno de los atributos es ajeno, señálelos con un círculo.
2. Recubrimiento minimal (escriba primero las dependencias funcionales y después las multivaluadas y de combinación si las hubiera).
3. Forma normal en la que se encuentra la relación.
4. Esquema relacional normalizado con las observaciones que estime pertinentes (si hubiera dependencias multivaluadas o de combinación no las tenga en cuenta a la hora de normalizar).

B.4. Dado el siguiente esquema de relación

$R < \{ O, R, U, V, W, X, Y, Z \},$
 $\{ XY \rightarrow Z, Z \rightarrow U, XYZ \rightarrow V, R \rightarrow X, X \rightarrow R, W \rightarrow O, O \rightarrow W \} >$

1. Hallar el cierre transitivo del descriptor XY.
2. Determinar las claves de R.
3. ¿En qué forma normal se encuentra R?
4. Descomponer R en un conjunto de relaciones en FNBC (indique las claves primarias de las relaciones resultantes).

B.5. Se desea diseñar una base de datos para una empresa de ventas que tiene representantes (R) en las distintas áreas (A) donde vende sus productos (P). Se supone que:

- Los representantes tienen un código (CR) y un conjunto de atributos (AR) . Análogamente, las áreas tienen CA y AA y los productos CP y AP.
- En cada A hay varios R y cada R trabaja en varias A.

- En todas las A se venden todos los P.
- Un R puede vender varios P y cada P se vende por varios R.
- Nunca dos R venden el mismo P en la misma A.
- Todo R vende el mismo conjunto de P en cada A donde trabaja.

Se pide:

1. Las dependencias que se deducen de cada uno de los supuestos del enunciado. Si de algún supuesto se deduce la no existencia de alguna dependencia indíquelo también.
2. Recubrimiento minimal.
3. Estructura relacional que considere más adecuada, analizando el nivel de normalización de cada una de las relaciones.

B.6. ¿En la siguiente extensión de relación se cumple algún tipo de dependencia? Si fuese así, indicarla y avanzar en el grado de normalización de la misma si ello fuese posible. Se supone que de cumplirse alguna dependencia en esta extensión se verificaría asimismo en cualquier otra extensión.

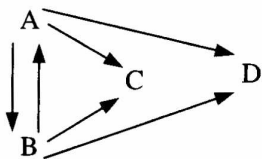
R

A	B	C
a1	b1	c1
a2	b1	c1
a1	b1	c2
a2	b1	c2
a1	b1	c3
a2	b1	c3
a1	b2	c3

B.7. ¿En que forma normal se encuentra la siguiente relación?:

IDIOMA	DNI_EMP	FAMILIAR
“Francés”	9999	“Pepe”
“Inglés”	9999	“Pepe”
“Francés”	9999	“María”
“Inglés”	9999	“María”
“Inglés”	9444	“Carmen”

B.8. Dado el grafo de la figura:



decir en qué forma normal se encuentra la relación $R(A, B, C, D)$.

B.9. Dada la relación siguiente, donde la clave primaria es (A, B, C) :

A	B	C

A1	B1	C2
A1	B2	C1
A2	B1	C1
A1	B1	C1

decir en qué forma normal se encuentra dicha extensión.

B.10. Dada la dependencia funcional:

$$\text{CÓD_E1}, \text{CÓD_E2} \rightarrow \text{CÓD_E3}$$

Si se cumple $\text{CÓD_E1}, \text{CÓD_E2} \not\rightarrow \text{CÓD_E3}$, (siendo CÓD_E2 el atributo extraño)

Determinar, mediante la aplicación de los axiomas de Armstrong, si la dependencia

$$\text{CÓD_E1}, \text{CÓD_E3} \rightarrow \text{CÓD_E2}$$

es completa o no completa y en este caso determinar cuál es el atributo extraño.

B.11. Dado el conjunto de dependencias:

$$\text{DF1} = B \rightarrow C, E \rightarrow B D, BC \rightarrow D, E \rightarrow A$$

$$\text{DF2} = E \rightarrow AB, B \rightarrow CD$$

si son o no equivalentes y, si no lo fuesen, explicar la razón.

B.12. Contestar cierto (C) o falso (F) a las siguientes preguntas:

() La relación $R(X, Y, Z)$ se puede descomponer sin pérdida en:

$R_1(X, Y)$

$R_2(X, Z)$

si, y sólo si, $X \rightarrow Y$ y $X \rightarrow Z$.

() El paso a la 3FN en proyecciones independientes nunca implica pérdida de dependencias.

() La relación $R(A, B, C, D; A \rightarrow B, B \rightarrow C, CD \rightarrow B)$ está normalizada.

() Dada la relación $R(A; D)$, si la normalizamos correctamente mediante análisis, no siempre obtendremos el mismo conjunto R_i de relaciones en 3FN.

() Para que un esquema relacional, que está en 3FN, se encuentre también en FNBC, es necesario y suficiente que existan clave candidatas solapadas.

() Para que un esquema de relación, que está en 3FN, se encuentre en 4FN, es necesario y suficiente que una de sus claves sea simple.

() Un esquema de relación binaria está siempre en 2FN.

() Una excesiva normalización puede complicar las consultas a la base de datos.

() Una relación binaria está siempre en 5FN.

B.13. Dado el esquema de relación $R(A,B,C,D,E)$ y los conjuntos de dependencias:

1: $A \rightarrow B, AB \rightarrow C, D \rightarrow AC, D \rightarrow E$

2: $A \rightarrow BC, D \rightarrow AE$

a) Son equivalentes.

b) No son equivalentes.

c) Son equivalentes según lo que signifiquen A, B, C, D y E .

B.14. Dado el siguiente esquema de relación:

$R < \{ O, R, U, V, W, X, Y, Z \},$
 $\{ XY \rightarrow Z, Z \rightarrow U, XYZ \rightarrow V, R \rightarrow X, X \rightarrow R, W \rightarrow O, O \rightarrow W \} >$

1. Hallar el cierre transitivo del descriptor XY.
2. Determinar las claves de R.
3. ¿En qué forma normal se encuentra R?
4. Descomponer R en un conjunto de relaciones en FNBC (indique las claves primarias de las relaciones resultantes).

SOLUCIONES A LOS EJERCICIOS PROPUESTOS

A. EJERCICIOS DE MODELADO

Se propone una posible solución para algunos de los ejercicios planteados. Debe tenerse en cuenta que esta solución no es única y que, en cualquier caso, puede ser discutida.

En cada ejercicio resuelto se presenta el esquema conceptual y el esquema relacional. Además, se incluirán, en algunos casos, las restricciones de borrado, así como alguna restricción de verificación o aserción. El objetivo no es estudiar todas las posibles restricciones de cada caso resuelto, sino mostrar ejemplos que puedan servir de guía al lector. Por ello, se discutirán aquellos casos que se han encontrado más relevantes.

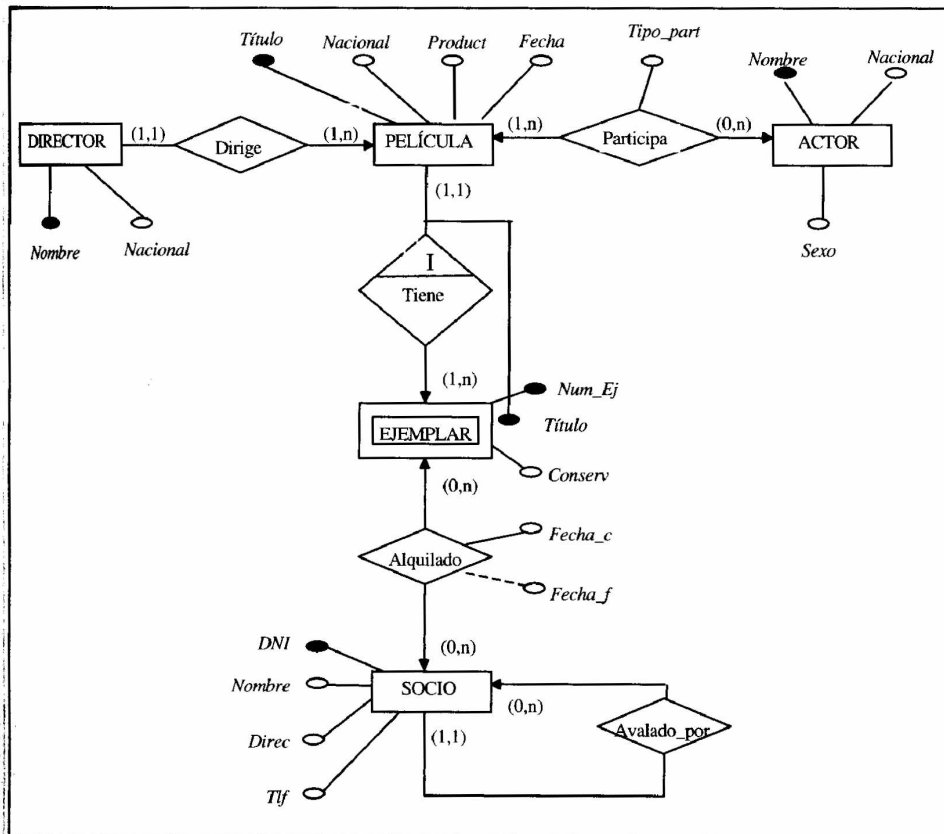
En cuanto a las transformaciones al relacional, se han realizado de modo que aparezcan las diversas posibilidades (una, dos o tres tablas). En algunos casos es posible que pudiera encontrarse acertada cualquier otra opción; no debe olvidarse que se presentan ejemplos planteados desde un punto de vista académico.

A.1. Base de datos para una cadena de Videoclubs

a) Esquema conceptual

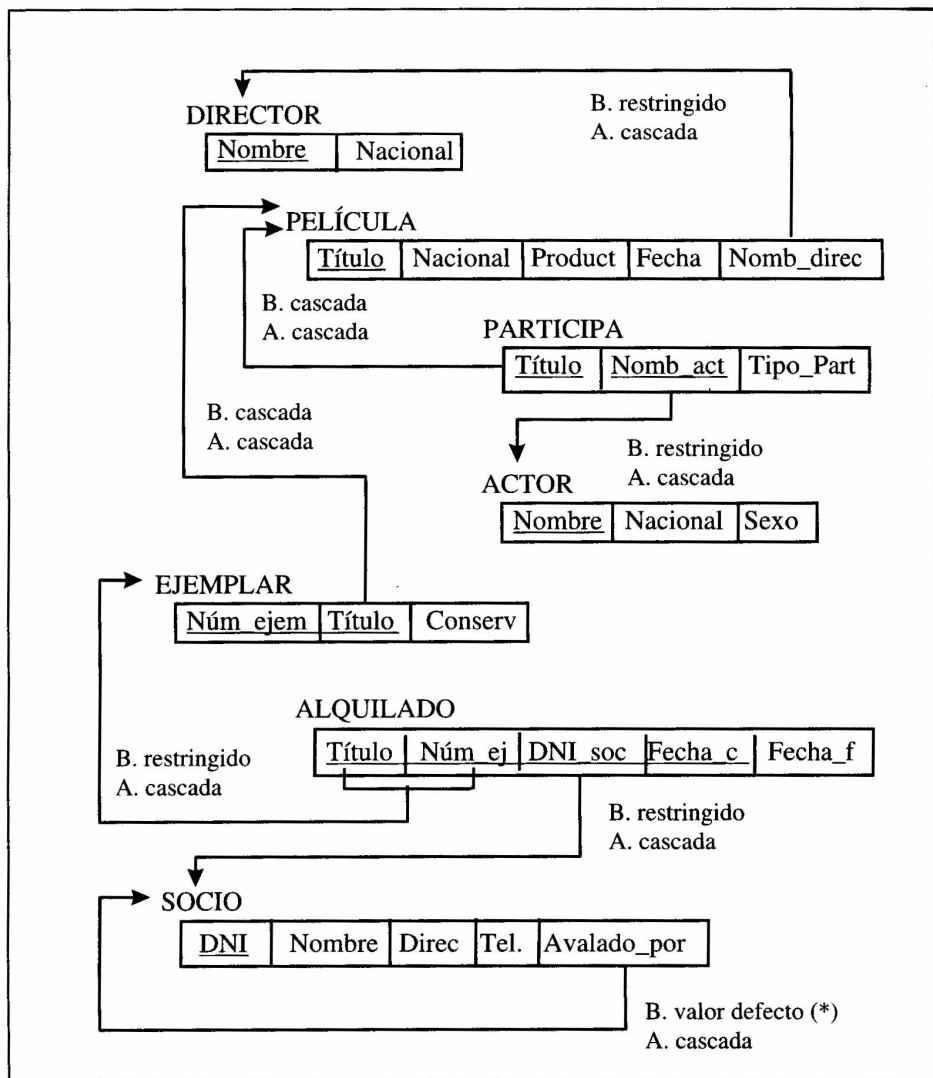
En el modelo E/R no se puede recoger la semántica de que, en un momento dado, un socio sólo pueda tener alquilados, como máximo, cuatro ejemplares. Tampoco se puede recoger el hecho de que un ejemplar sólo pueda estar alquilado, en un momento

determinado, a un socio. El modo de recoger esta semántica sería crear una interrelación "Tiene alquilado" que representaría los ejemplares que en un momento dado tiene alquilados un determinado socio, y mantener la interrelación "Alquilado" como el histórico.



b) Esquema relacional

(*) Si un socio decide darse de baja del vídeo club se le debe permitir, aún en el caso de que éste haya avalado a algún otro socio, por lo que no podría hacerse un borrado restringido; tampoco tiene sentido que se eliminen todos los socios a los que éste haya avalado, por lo que tampoco sería un borrado en cascada. Por otra parte, dado que un socio siempre tiene que ser avalado por otro, tampoco se debe permitir poner un nulo en el campo *Avalado_por*. Por tanto, la opción que parece más lógica es la de poner un valor por defecto (por ejemplo, el gerente del videoclub, que sería el primer socio de la base de datos).



Otro modo de resolver este problema, sería añadir un campo, *fecha_baja*, a la tabla socio que indique cuándo un socio se ha dado de baja en el videoclub. En este caso, el borrado sería restringido. Eliminar un socio de la base de datos implicaría modificar el valor de este atributo. De este modo, se mantendrían los datos de todos los socios avalistas, incluso aunque éstos se hubieran dado de baja. Teóricamente, esto se resolvería mediante disparadores, el problema es que en la práctica, aun cuando los productos disponen de disparadores, las facilidades que éstos ofrecen pueden ser restringidas.

En cualquier caso, la decisión final dependerá siempre de la semántica que el usuario quiera recoger en su base de datos. Si se decide la última opción planteada, podría hacerse de modo transparente para quién estuviera actualizando la base de datos; así, el campo *fecha_baja* sería nulo por defecto (en la creación de un socio) y se definiría un disparador que se encargara de modificar este valor cuando se intentara eliminar un socio de la base de datos.

Este disparador podría ser:

```
CREATE TRIGGER borrar_socio
  INSTEAD OF DELETE ON socio
  FOR EACH ROW
  (UPDATE socio
    SET fecha_baja=fecha_actual)
```

- La restricción de que un socio no puede tener alquilados más de cuatro ejemplares simultáneamente, deberá recogerse mediante una restricción de verificación definida en la tabla ALQUILADO:

```
CREATE TABLE alquilado(
.....
```

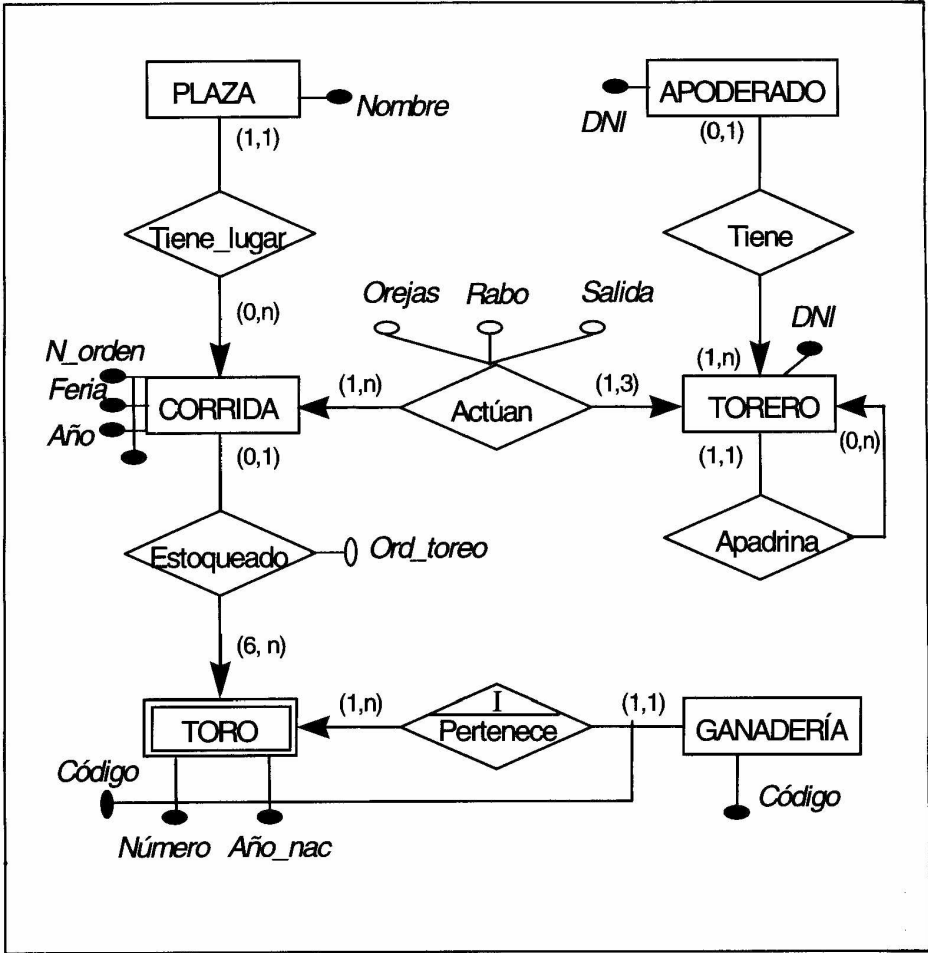
```
CHECK NOT EXIST (SELECT COUNT(*)
                  FROM alquilado
                  WHERE fecha_f=NULL
                  GROUP BY DNI_socio
                  HAVING COUNT(*)>4));
```

Si se desea controlar que, mientras que un ejemplar está alquilado a un socio no pueda estar alquilado a otro, añadiríamos la siguiente restricción de verificación a la tabla ALQUILADO:

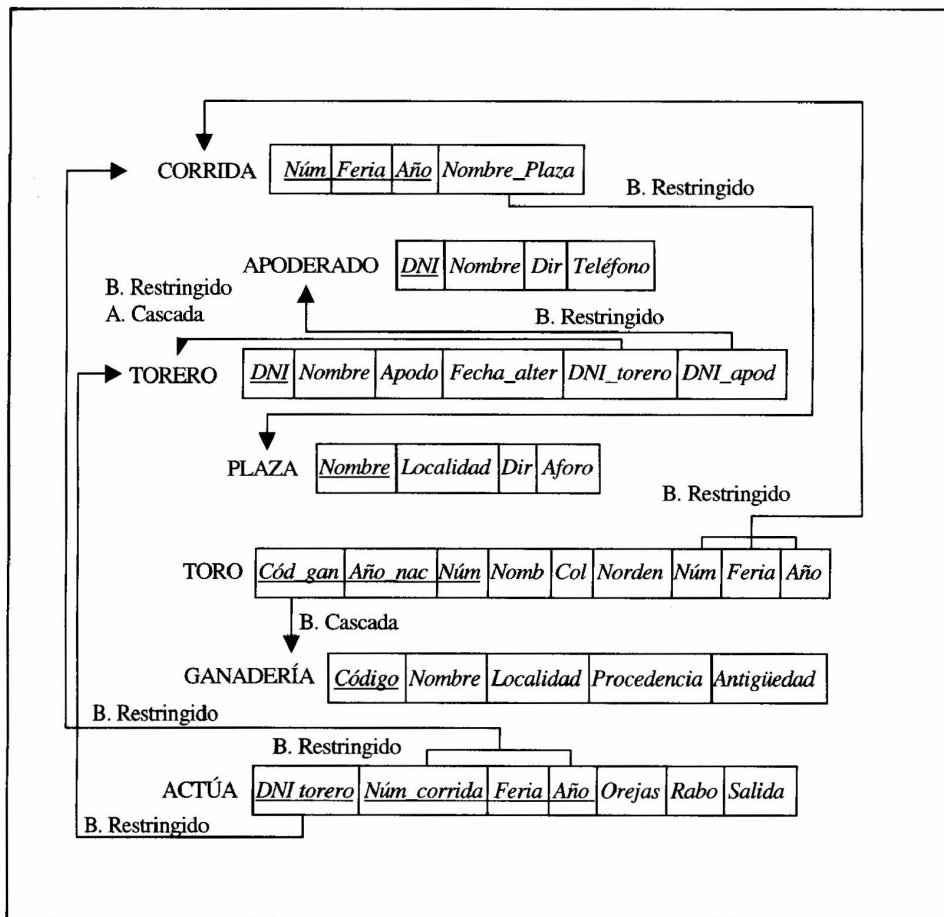
```
CHECK NOT EXIST (SELECT COUNT(*)
                  FROM alquilado
                  WHERE fecha_f=NULL
                  GROUP BY (Título, Num_ejem)
                  HAVING COUNT(*)>1));
```

A.2. Base de datos para corridas de toros

a) Esquema conceptual



b) Esquema relacional



- La restricción de que en una corrida actúan como máximo tres toreros, deberá recogerse con una restricción de verificación definida sobre la tabla ACTÚA:

```
CREATE TABLE actua(
```

```
.....
```

```
CHECK NOT EXIST
```

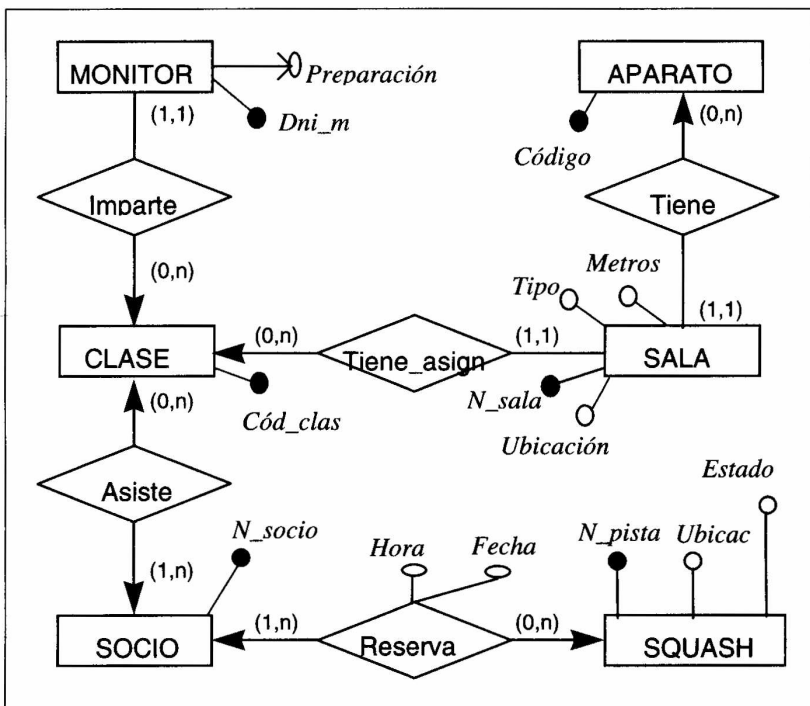
```
(SELECT Num_corrida, Feria, Año, COUNT(DNI_torero)
FROM actua
GROUP BY (Núm_corrida, Feria, Año)
HAVING COUNT(DNI_torero)>3));
```

- La restricción de que en una corrida se estoquean como mínimo seis toros, deberá recogerse con una restricción de verificación definida sobre la tabla TORO:

```
CREATE TABLE toro(
.....
CHECK NOT EXIST
(SELECT Núm_corrida, Feria, Año, COUNT(*)
FROM toro
GROUP BY (Núm_corrida, Feria, Año)
HAVING COUNT(*)<6);
```

A.4. Base de datos para un gimnasio (G1)

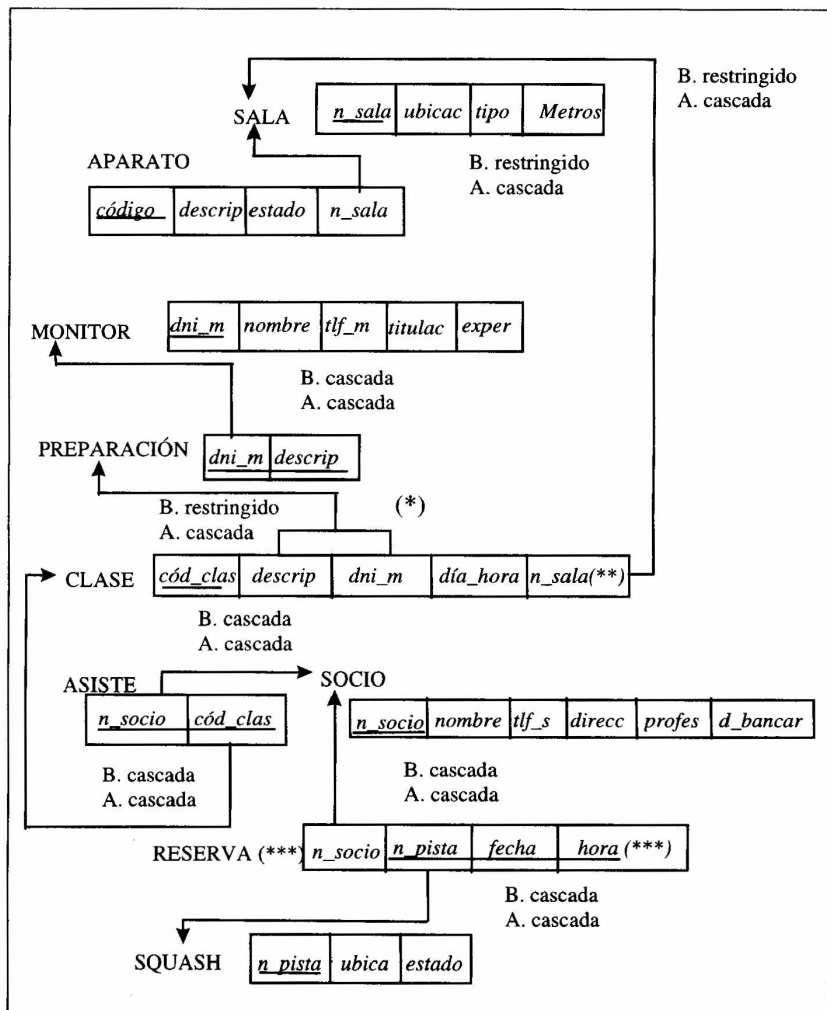
a) Esquema conceptual



Existe una restricción que no se puede recoger en este esquema: una determinada pista, en una cierta fecha y hora sólo la puede reservar un socio. Para recogerla se puede, al igual que en el ejercicio A.1, introducir dos interrelaciones, una para las reservas actuales y otra para el histórico. Otra posibilidad es modelar "Reserva" como

una entidad débil respecto de SQUASH con una dependencia en identificación, de modo que el atributo identificador principal (AIP) de "Reserva" estuviera compuesto por *N_pista* (de SQUASH), *fecha* y *hora*.

b) Esquema relacional



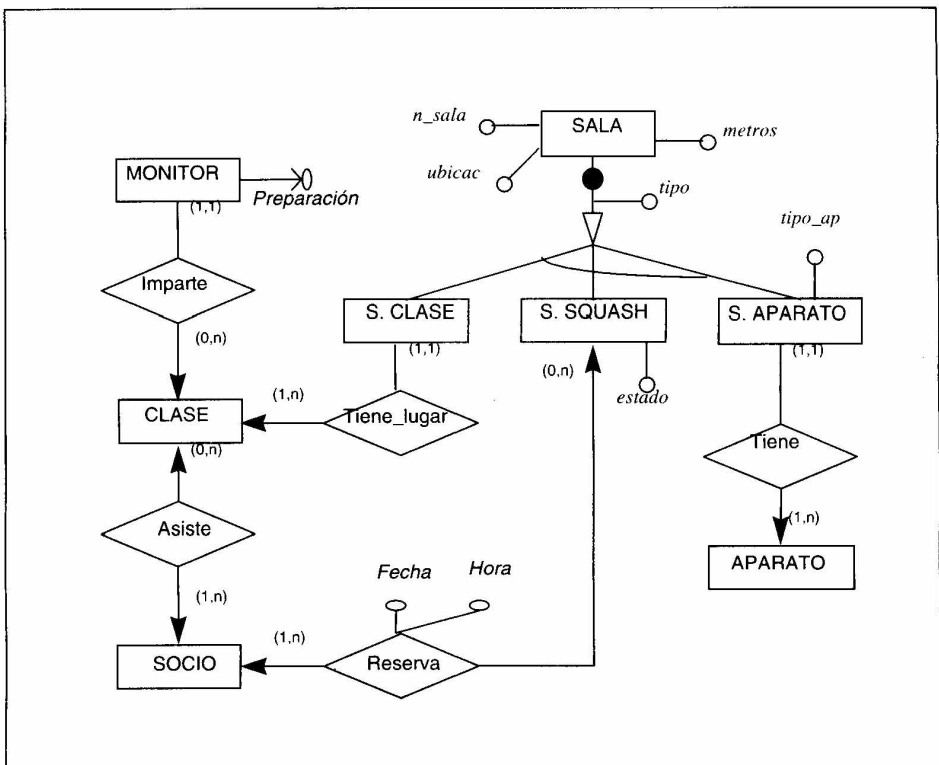
(*) *dni_m* es clave ajena de monitor. Sin embargo, si definimos (*dni_m*, *descrip*) como clave ajena de la tabla CLAS que referencia a la tabla PREPARACIÓN, mantenemos esta misma restricción y además podemos controlar que un monitor sólo imparta clases para las que esté preparado.

(**) Este atributo tiene la opción NOT NULL, ya que una clase siempre tiene que impartirse en una SALA.

(***) Se puede observar que N_socio no forma parte de la clave primaria de RESERVA, ya que dado un N_pista , $Fecha$ y $Hora$ sólo puede haber un socio que haga la reserva.

A.5. Base de datos para un gimnasio (G2)

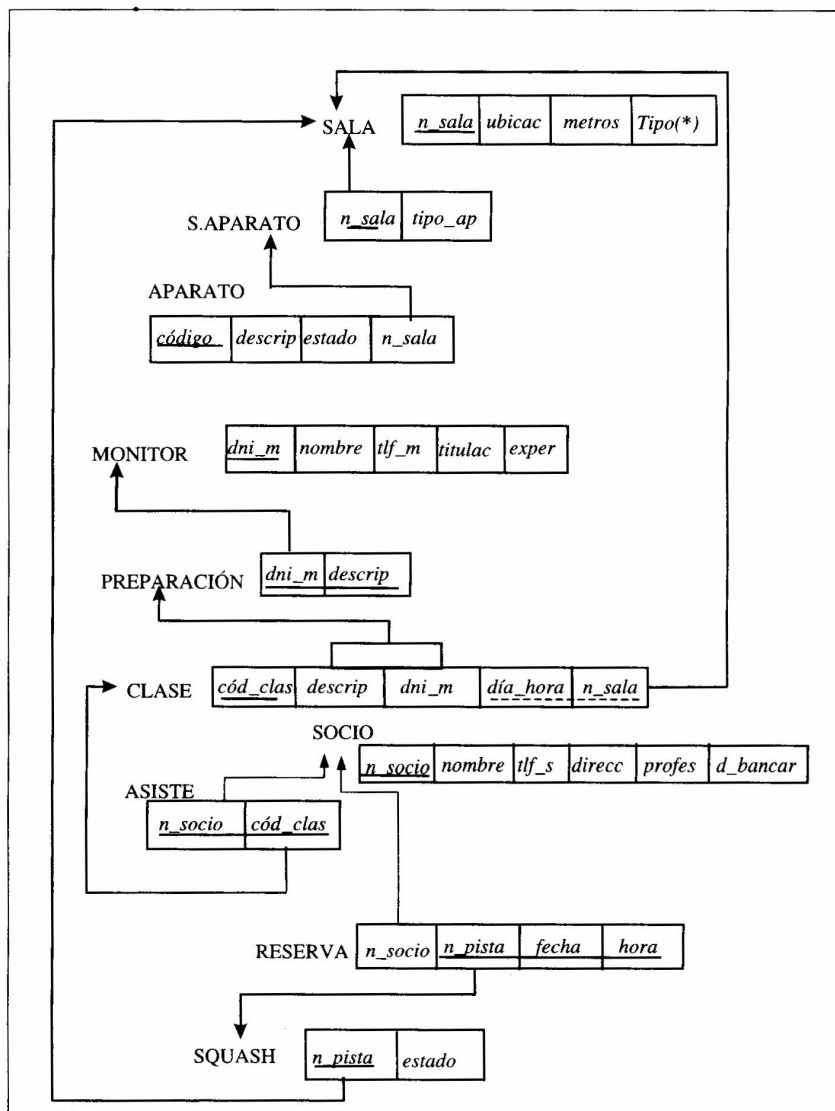
a) Esquema conceptual



El atributo *Tipo_ap* que aparece en la entidad S. APARATO corresponde a atributo *tipo* de la entidad SALA del ejercicio A.4. Este atributo representa el tipo de sala dependiendo de los aparatos que tenga (cardio, muscular, general). En el ejercicio A.4 no se diferenciaba entre salas de clase y de aparatos, por lo que dicho atributo correspondía a la entidad SALA. Sin embargo, ahora existe una entidad específica para las salas de aparatos, por lo que dicho atributo corresponde a esta entidad.

Tipo, en la jerarquía, representa el tipo de sala, según sea de Clase, Squash o Aparato

b) Esquema relacional



(*) Se definirá un dominio para el atributo Tipo de la tabla SALA, que indique si se trata de una sala de Clase, de Squash o de Aparatos. Además, dicho atributo deberá definirse como no nulo (NOT NULL) para reflejar la totalidad de la jerarquía:

```
CREATE DOMAIN tipo_sala AS CHAR (1)
CHECK VALUES IN ('C', 'S', 'A')
```

La exclusividad de la jerarquía se puede controlar mediante aserciones. Obsérvese que la tabla correspondiente al subtipo S.CLASE no aparece, ya que dicho subtipo no tiene atributos propios; sin embargo, se debería controlar mediante otra aserción, que al atributo *N_sala* de CLASE le corresponde en SALA un valor "C" en el atributo *Tipo*.

Tendríamos, por tanto, las siguientes restricciones:

1. Una aserción que verifique que las clases sólo se imparten en salas de clase.

```
CREATE ASSERTION clase_sala
CHECK (( SELECT n_sala
        FROM clase) IN
        (SELECT n_sala
        FROM sala
        WHERE tipo='C'));
```

2. Una aserción que verifique que las pistas de squash son salas de este tipo.

```
CREATE ASSERTION squash_sala

CHECK (( SELECT n_pista
        FROM squash) IN
        (SELECT n_sala
        FROM sala
        WHERE tipo='S'));
```

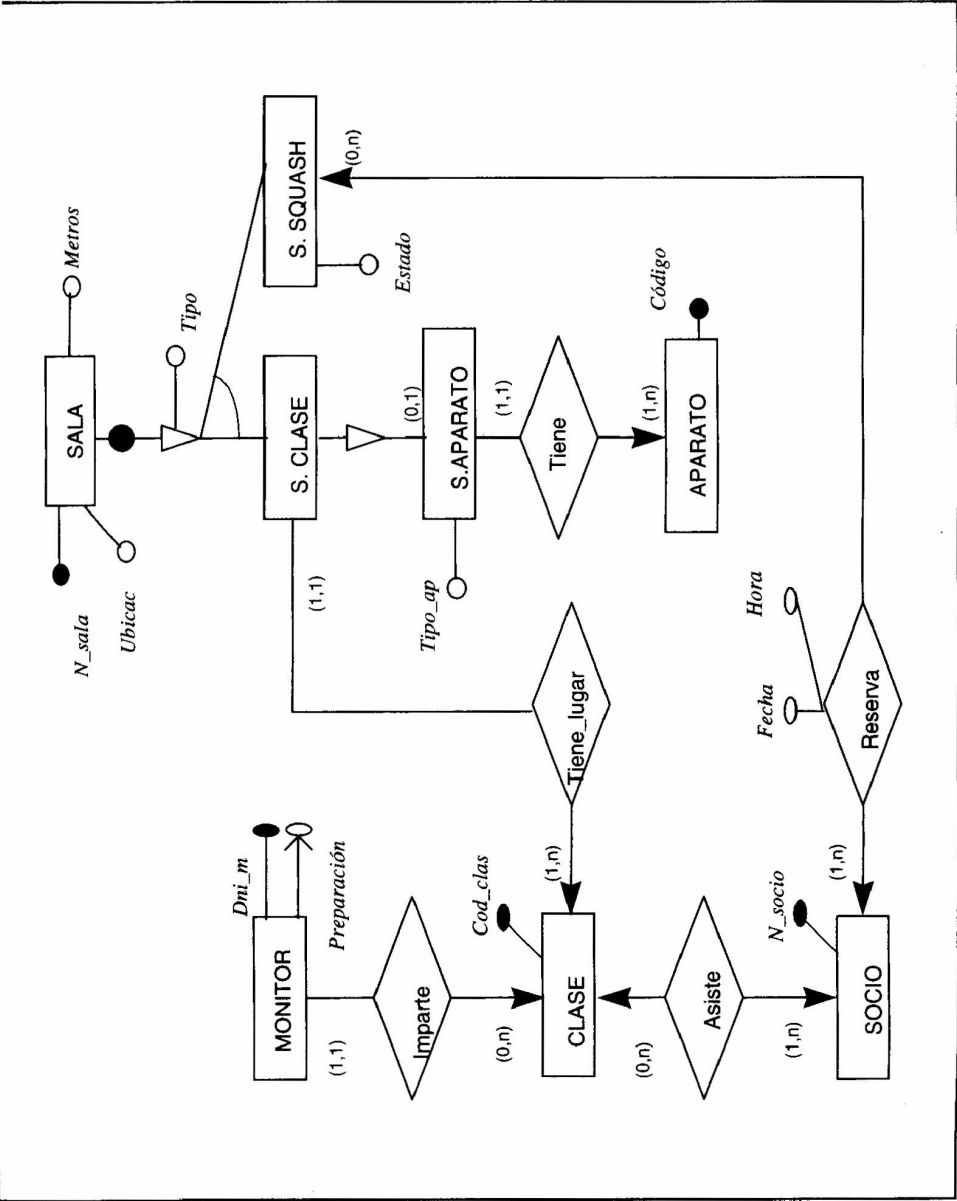
3. Una aserción que verifique que las salas de aparatos no son salas de clase ni de squash.

```
CREATE ASSERTION s_aparato_sala
CHECK (( SELECT n_sala
        FROM s_aparato) IN
        (SELECT n_sala
        FROM sala
        WHERE tipo='A'));
```

Las aserciones anteriores, junto con la restricción de dominio del atributo *Tipo* de SALA, garantizan la exclusividad de la jerarquía.

A.6. Base de datos para un gimnasio (G3)

a) Esquema conceptual



b) Esquema relacional

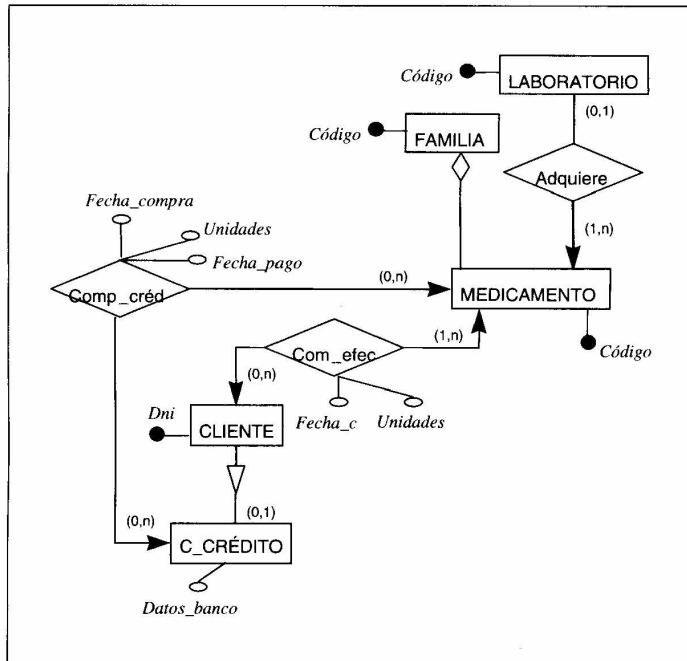
El esquema relacional podría ser el mismo que el propuesto para el ejercicio A.5, pero cambiarían algunas restricciones:

- Se mantienen las restricciones 2 y 3.
- Se modifica la restricción 1 (ahora, en las salas de aparatos también se pueden impartir clases)

```
CREATE ASSERTION clase_sala
CHECK (( SELECT n_sala
        FROM clase) IN
        (SELECT n_sala
        FROM sala
        WHERE tipo IN ('C', 'A')));
```

A.7. Base de datos para una farmacia

a) Esquema conceptual




```
CREATE TRIGGER actualizar_vendidas1
AFTER INSERT ON c_crédito
FOR EACH ROW
( UPDATE medicamento
  SET vendas=vendas + unidades
  WHERE código=cod_med)
```

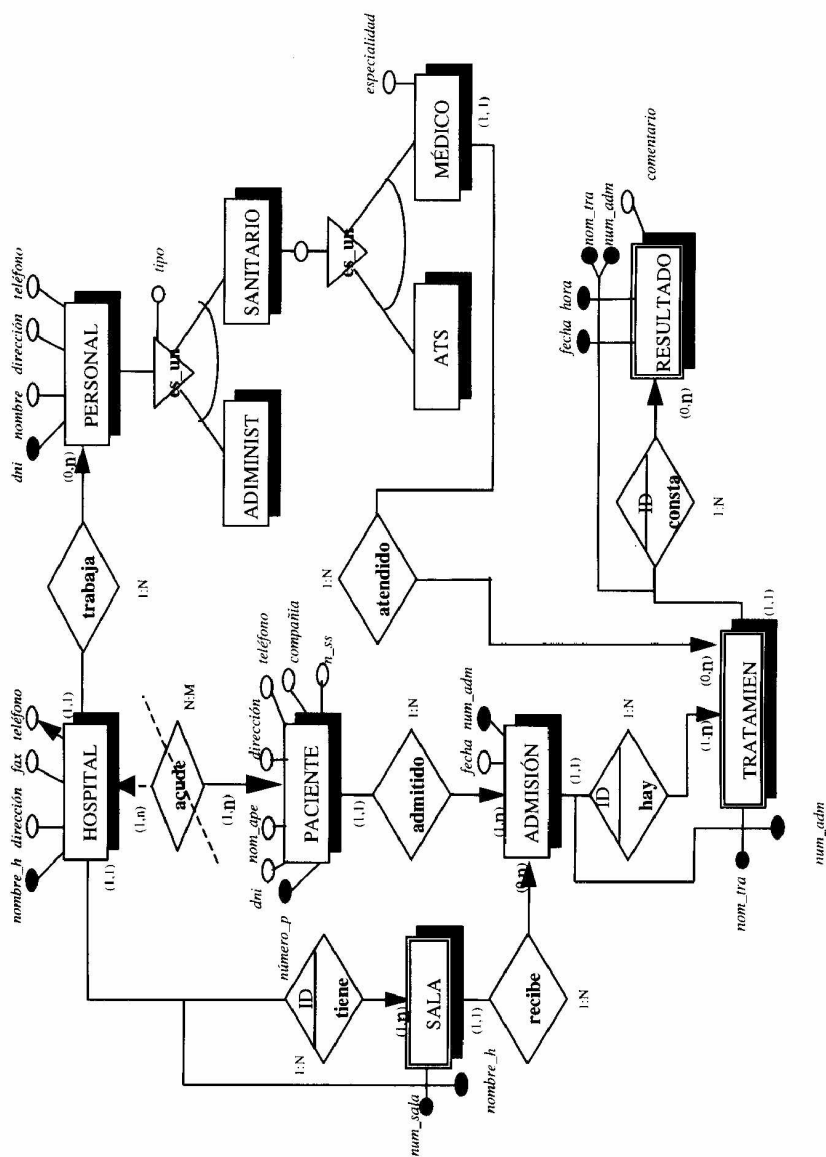
```
CREATE TRIGGER actualizar_vendidas2
AFTER INSERT ON comp_efect
FOR EACH ROW
( UPDATE medicamento
  SET vendas=vendas + unidades
  WHERE código=cod_med)
```

A.9. Base de datos para un consorcio de hospitales

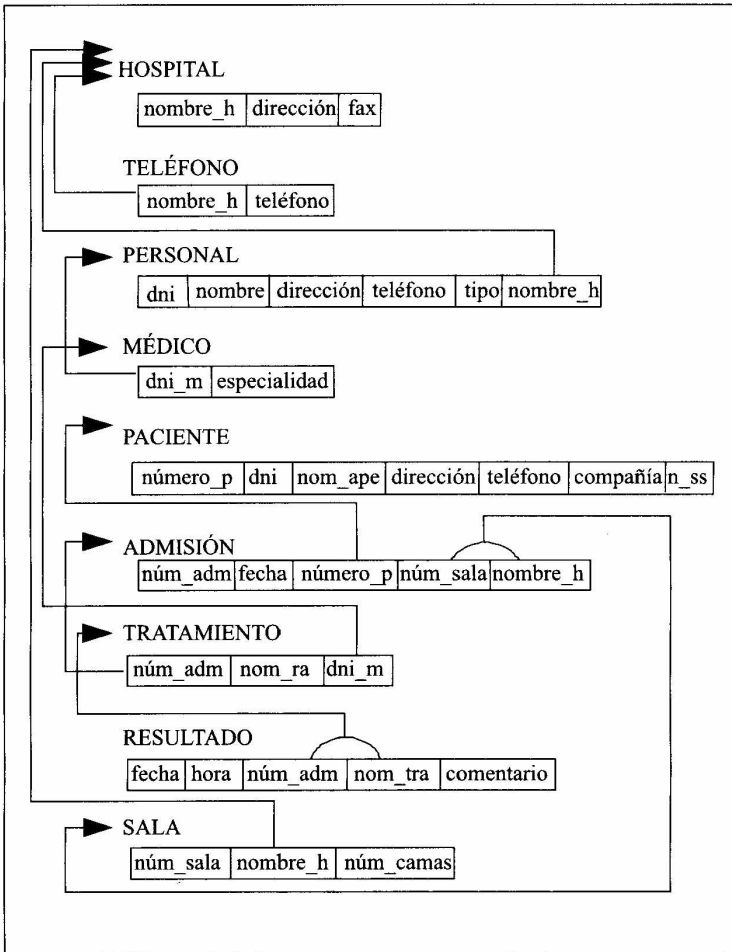
a) Esquema conceptual

La interrelación acude es redundante, por lo que se eliminará en el esquema relacional.

La generalización de PERSONAL no es total ya que se considera que existe otro tipo de personal además del administrativo y del de servicios (por ejemplo, personal de limpieza).



b) Esquema relacional



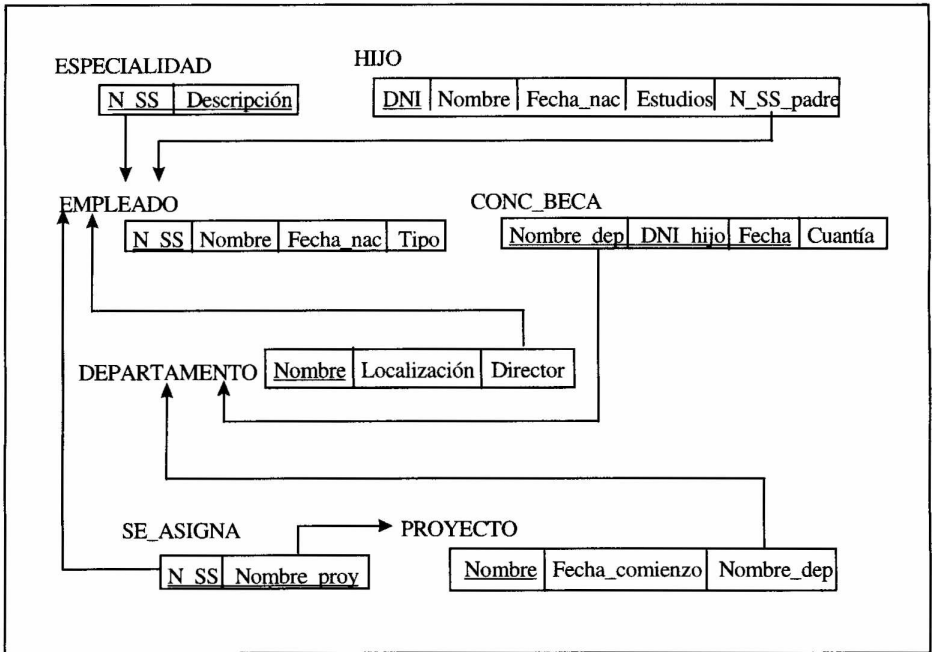
La clave primaria de la tabla TELÉFONO es *teléfono*, ya que los teléfonos no pueden repetirse.

De los subtipos de la generalización sólo se mantiene MÉDICO como tabla, por ser el único subtipo que tiene atributos e interrelaciones propias. El resto del personal se distingue mediante el atributo *tipo* que tomará el valor nulo cuando se trate de personal que no sea ni administrativo ni sanitario (se definiría un dominio para este atributo):

```
CREATE DOMAIN tipo_personal AS CHAR (3)
CHECK VALUES IN ('ADM', 'ATS', 'MED')
```


Aparece un ciclo entre DEPARTAMENTO INGENIERO y PROYECTO. Sin embargo, no es necesario comprobar la redundancia del mismo, puesto que, en principio, un ingeniero puede estar asignado a un proyecto controlado por un departamento distinto al que pertenece. Por lo tanto, no hay redundancia.

b) Esquema relacional



La clave primaria de CONC_BECA incluye también el atributo *fecha* debido a que un mismo departamento podría conceder más de una beca, en distintas fechas, al mismo hijo.

En este caso, la generalización se ha implementado en una única tabla. El modo de saber si un empleado es Director, representante de Ventas o Ingeniero es mediante el atributo *Tipo* que tiene que ser no nulo (NOT NULL) para reflejar la totalidad de la generalización; se definiría un dominio para este atributo:

```
CREATE DOMAIN tipo_empleado AS CHAR (1)
CHECK VALUES IN ('D', 'V', 'I')
```


Además, se definirán tres aserciones:

1. Una aserción que verifique que los directores de departamento son empleados de tipo director:

```
CREATE ASSERTION departamento_empleado
CHECK (( SELECT director
        FROM departamento) IN
        (SELECT N_SS
        FROM empleado
        WHERE tipo='D'));
```

2. Una aserción que verifique que todo proyecto se le asigna a un ingeniero:

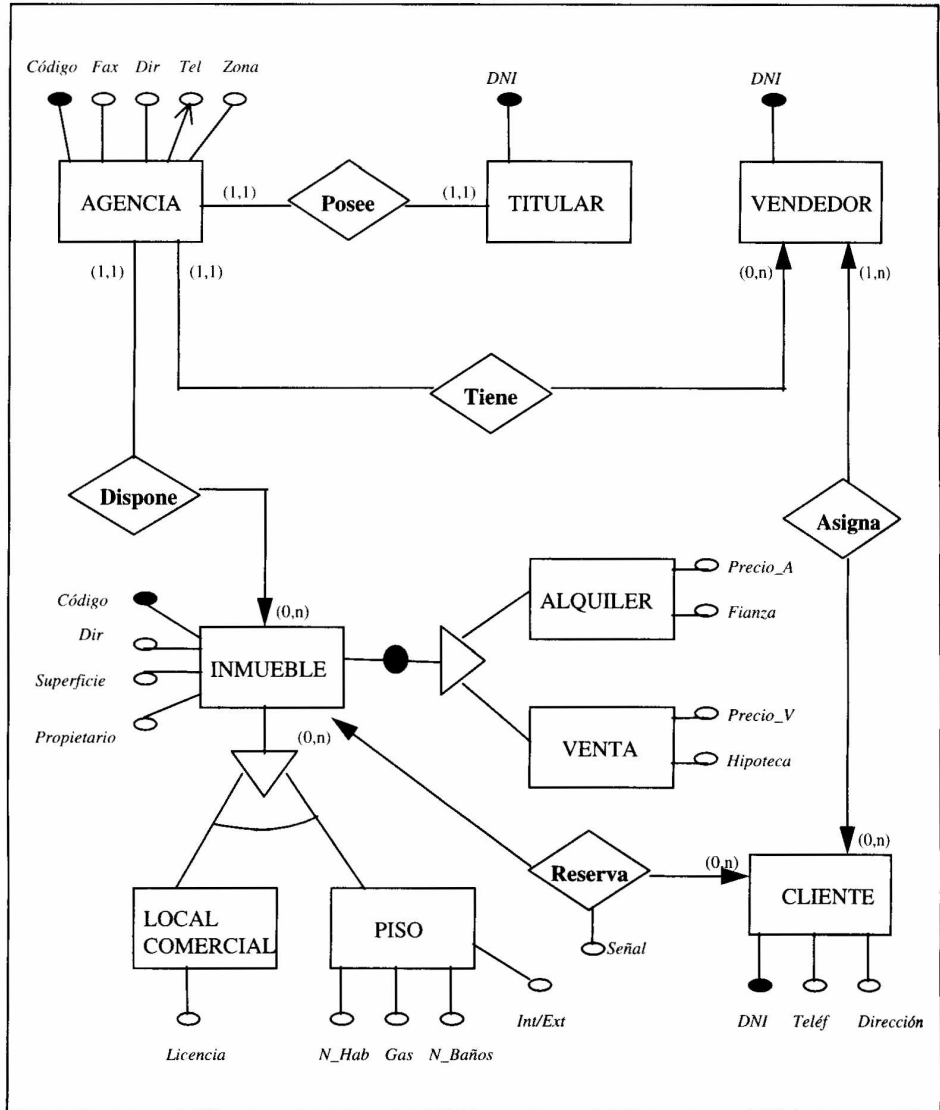
```
CREATE ASSERTION asigna_empleado
CHECK (( SELECT N_SS
        FROM se_asigna) IN
        (SELECT N_SS
        FROM empleado
        WHERE tipo='I'));
```

3. Una aserción que verifique que las especialidades son siempre de ingenieros:

```
CREATE ASSERTION especialidad_empleado
CHECK (( SELECT N_SS
        FROM especialidad) IN
        (SELECT N_SS
        FROM empleado
        WHERE tipo='I'));
```

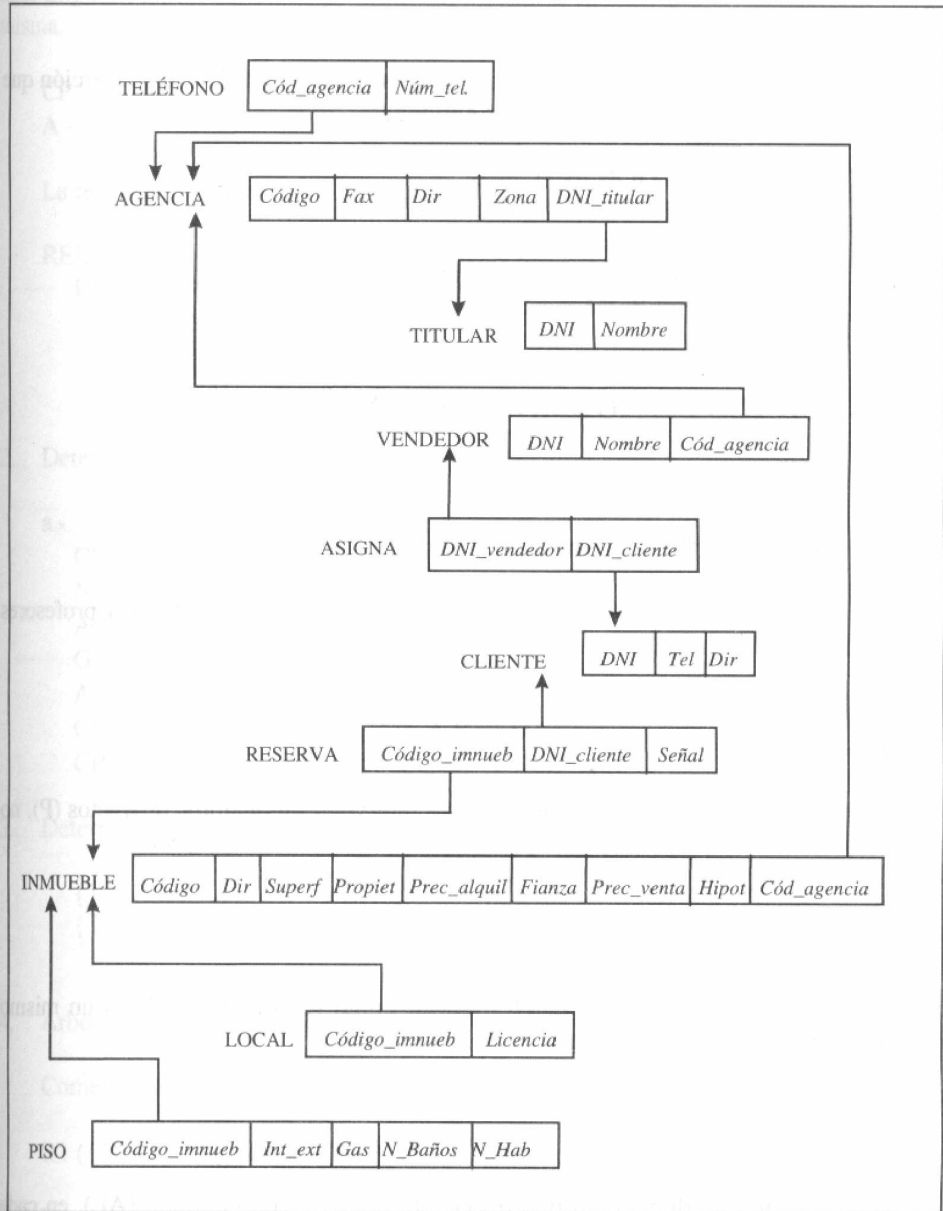
A.12. Base de datos para una red de agencias inmobiliarias

a) Esquema conceptual



Aparece un ciclo entre AGENCIA-VENDEDOR-CLIENTE e INMUEBLE. Sin embargo, puede comprobarse que ninguna de las interrelaciones que forman parte de este ciclo es redundante.

b) Esquema relacional



La clave primaria de la tabla **TELÉFONO** es *núm_tel*, ya que el número de teléfono no puede repetirse

La tabla inmueble recoge tanto los inmuebles en alquiler como los que están en venta. Debido a que se trata de una generalización total y solapada no es necesario un atributo discriminante ni restricciones asociadas.

En cuanto a los locales comerciales o pisos, es necesario definir una aserción que implemente la exclusividad entre estas dos entidades:

```
CREATE ASSERTION local_piso
CHECK (( SELECT Código_inmueble
        FROM piso) NOT IN
        (SELECT Código_inmueble
        FROM local));
```

B. EJERCICIOS DE NORMALIZACIÓN

B.2

1. Determinamos las dependencias funcionales:

Un profesor se identifica por un código de profesor (CP) y todos los profesores tienen nombres (NP) distintos.

$CP \rightarrow NP$
 $NP \rightarrow CP$

Un profesor puede tener varios títulos (T) e intervenir en distintos proyectos (P), no existiendo ningún título determinado para intervenir en un cierto proyecto.

$CP \twoheadrightarrow T \mid P$
 $T \not\rightarrow P$

Cada asignatura (A) tiene un único profesor como responsable, si bien un mismo profesor puede ser responsable de más de una asignatura.

$A \rightarrow CP$
 $CP \not\rightarrow A$

Las asignaturas se dividen en uno o más grupos (G). Todo alumno (AL), en cada asignatura, pertenece a un único grupo.

$G \rightarrow A$
 $AL, A \rightarrow G$

Cada profesor depende siempre y únicamente de un departamento (D). Así mismo, toda asignatura está ligada a un único departamento, el del profesor responsable de la misma.

$CP \rightarrow D$

$A \rightarrow D$

La relación resultante será la siguiente:

RELACION ({CP, NP, T, P, A, G, AL, D},

DF={ $CP \rightarrow NP$, $NP \rightarrow CP$, $A \rightarrow CP$, $G \rightarrow A$, $(AL, A) \rightarrow G$, $CP \rightarrow D$,
 $A \rightarrow D$ }

DM={ $CP \rightarrow \rightarrow T \mid P$ }

DC={ $\emptyset$ })

2. Determinamos el recubrimiento minimal:

a.-

$CP \rightarrow NP$

$NP \rightarrow CP$

$A \rightarrow CP$

$G \rightarrow A$

$AL, A \rightarrow G$

$CP \rightarrow D$

$CP \rightarrow \rightarrow T \mid P$

b.-

$CP \rightarrow NP$

$NP \rightarrow CP$

$G \rightarrow A$

$AL, A \rightarrow G$

$G \rightarrow CP$

$CP \rightarrow D$

$CP \rightarrow T \mid D$

3. Determinamos las claves de la relación:

{T, P, A, AL}

{T, P, G, AL}

4. Árbol de descomposición:

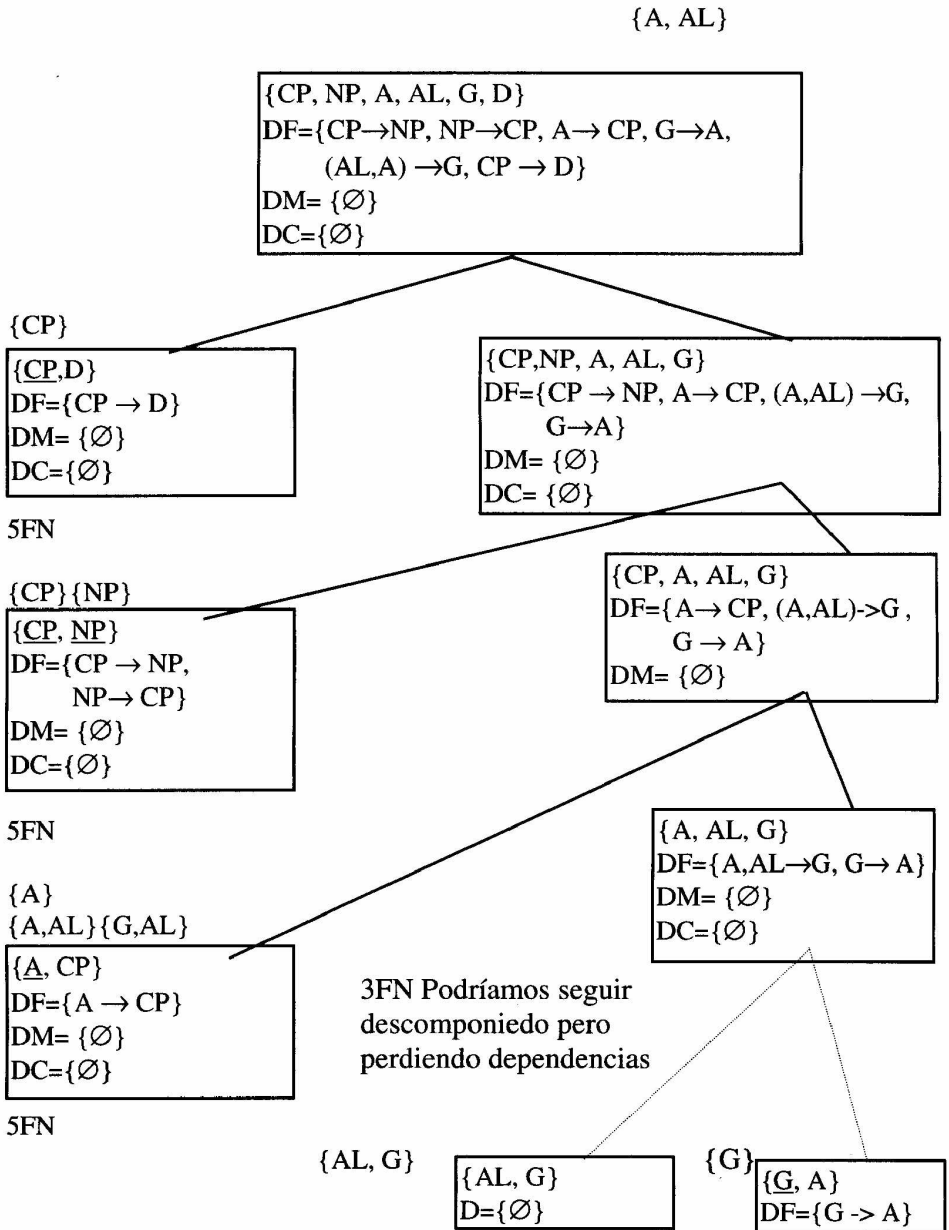
Comenzamos a descomponer a partir de las dependencias multivaluadas:

R1 ({CP, T, P}, { $CP \rightarrow \rightarrow T \mid P$ }) La relación está en FNBC pero no en 4FN.

Descomponemos

R11({CP, T} D={ $\emptyset$ }) 5FN

R12({CP, P} D={ $\emptyset$ }) 5FN



B.3

1. a) $H, E \rightarrow Au$; $H, P \rightarrow Au$;

b) $H, Au \rightarrow As$

c) $D \rightarrow Tl$

d) $H, E \rightarrow As$

e) $H \rightarrow As \mid G; As \rightarrow G \mid T$

f) $E, As \rightarrow N$

g) $P \rightarrow Tl; P \rightarrow D; Tl \rightarrow P; D \rightarrow P; D \rightarrow Tl$

h) $P \rightarrow G \mid T$

2. $H, E \rightarrow Au; H, P \rightarrow Au; H, Au \rightarrow As; E, As \rightarrow N$

$P \rightarrow D; P \rightarrow Tl; Tl \rightarrow P; D \rightarrow P;$

$H \rightarrow As \mid G; As \rightarrow G \mid T; P \rightarrow G \mid T$

(Las dependencias no redundantes entre profesor, despacho y teléfono pueden ser otras).

3. CLAVES:

P, D, Tl son atributos equivalentes; dejamos sólo P

IMPLICANTES: H, E, P, Au, As

IMPLICADOS : Au, As, N

Atributos que no se encuentran en ninguna dependencia funcional: G, T

Hay tres claves: H, E, P, G, T; H, E, D, G, T; H, E, Tl, G, T

Atributos no principales: N, As, Au

N no depende de la totalidad de la clave, luego la relación no está en 2FN

4. Dado que nos dicen que no se tengan en cuenta ni las dependencias multivaluadas ni las de combinación, se puede aplicar el método de síntesis (lo que nos permite asegurar que las relaciones resultantes están, al menos, en 3FN)

R1 (H, E, Au; H, E $\rightarrow$ Au)

* Aula por estudiante y hora*

R2 (H, P, Au; H, P $\rightarrow$ Au)

* Aula por profesor y hora*

R3 (H, Au, As; H, Au $\rightarrow$ As)

* Asignatura por aula y hora *

R3 (H, Au, As; H, Au $\rightarrow$ As)

* Asignatura por aula y hora *

R4 (<u>E</u> ,As,N; E, As $\rightarrow$ N)	* Nota por asignatura y estudiante *
R5 (<u>P</u> , <u>D</u> , <u>TI</u> ; P $\leftrightarrow$ D; P $\leftrightarrow$ TI)	* Profesor con despacho y teléfono*
R6 (<u>H</u> , <u>E</u> , <u>P</u> , <u>G</u> , <u>T</u> ; P $\rightarrow\rightarrow$ G $\mid$ T)	* La clave de la relación*

Observaciones: Las relaciones R1 a R5 están en 5FN. La relación R6 no está en 4FN debido a la dependencia multivaluada: P $\rightarrow\rightarrow$ G $\mid$ T. Si la sustituyéramos por R61 (P, G, T; P $\rightarrow\rightarrow$ G $\mid$ T), y normalizáramos, tendríamos:

R611 (P,G)

R612 (P,T)

Nos quedarían las siguientes relaciones (todas ellas en 5FN):

R1, R2, R3, R4, R5, R611, R612

B.4

1. R, U, V, X, Y, Z
2. Hay cuatro claves : { R Y W }, { R Y O }, { X Y W } y { X Y O }
3. En primera forma normal, ya que existen atributos no principales, como U, que tienen dependencia funcional no completa respecto de cada una de las claves.
4. R1 < { X Y Z }, { X Y $\rightarrow$ Z } >

R2 < { Z U }, { Z $\rightarrow$ U } >

R3 < { Z V }, { Z $\rightarrow$ V } >

R4 < { W O }, { W $\rightarrow$ O, O $\rightarrow$ W } >

R5 < { R X }, { R $\rightarrow$ X, X $\rightarrow$ R } >

R6 < { X Y W }, { } >

B.5

1. Las dependencias son las siguientes:

a) $CR \rightarrow AR ; CA \rightarrow AA ; CP \rightarrow AP$

b) $CA \not\rightarrow CR ; CR \not\rightarrow CA$

c) $CA \not\rightarrow CP ; CP \not\rightarrow CA$

d) $CR \not\rightarrow CP ; CP \not\rightarrow CR$

e) $CP, CA \rightarrow CR$

f) $CR \rightarrow\rightarrow CP \mid CA$

2. $CR \rightarrow AR ; CA \rightarrow AA ; CP \rightarrow AP$

$CP, CA \rightarrow CR ; CR \rightarrow P \mid A$

3.

R (CR, AR; $CR \rightarrow AR$) 5FN*

A (CA, AA; $CA \rightarrow AA$) 5FN*

P (CP, AP; $CP \rightarrow AP$) 5FN*

* Suponemos que entre los atributos de las tres relaciones no existen dependencias.

RAP (CP, CA, CR; $CP, CA \rightarrow CR, CR \rightarrow\rightarrow CP \mid CA$) FNBC

Se encuentra en FNBC pero no en 4FN debido a la dependencia multivaluada.

Se podría considerar alternatively otra estructura:

RAP1 (CR, CP) 5FN

RAP2 (CR, CA) 5FN

En este caso las relaciones se encuentran en 5FN, pero se pierde la dependencia:

$CR, CA \rightarrow CR$, la cual debería controlarse mediante un disparador.

Consideramos preferible la primera estructura.

B.7. Está en tercera forma normal

B.8. Está en tercera forma normal.

B.9. Se encuentra en 4FN

B.10. Es incompleta, siendo COD_E3 un atributo extraño

B.11. En la dependencia $BC \rightarrow D$ de DF1, el atributo B es redundante, quedando por tanto reducida a $C \rightarrow D$, la cual no se puede inferir de DF2. Por tanto:

Los dos conjuntos de dependencias DF1 y DF2 no son equivalentes, ya que la dependencia $C \rightarrow D$ de DF1 no se encuentra ni se puede deducir de DF2

B.12. La respuesta correcta es la a), son equivalentes.

EJEMPLO COMPLETO

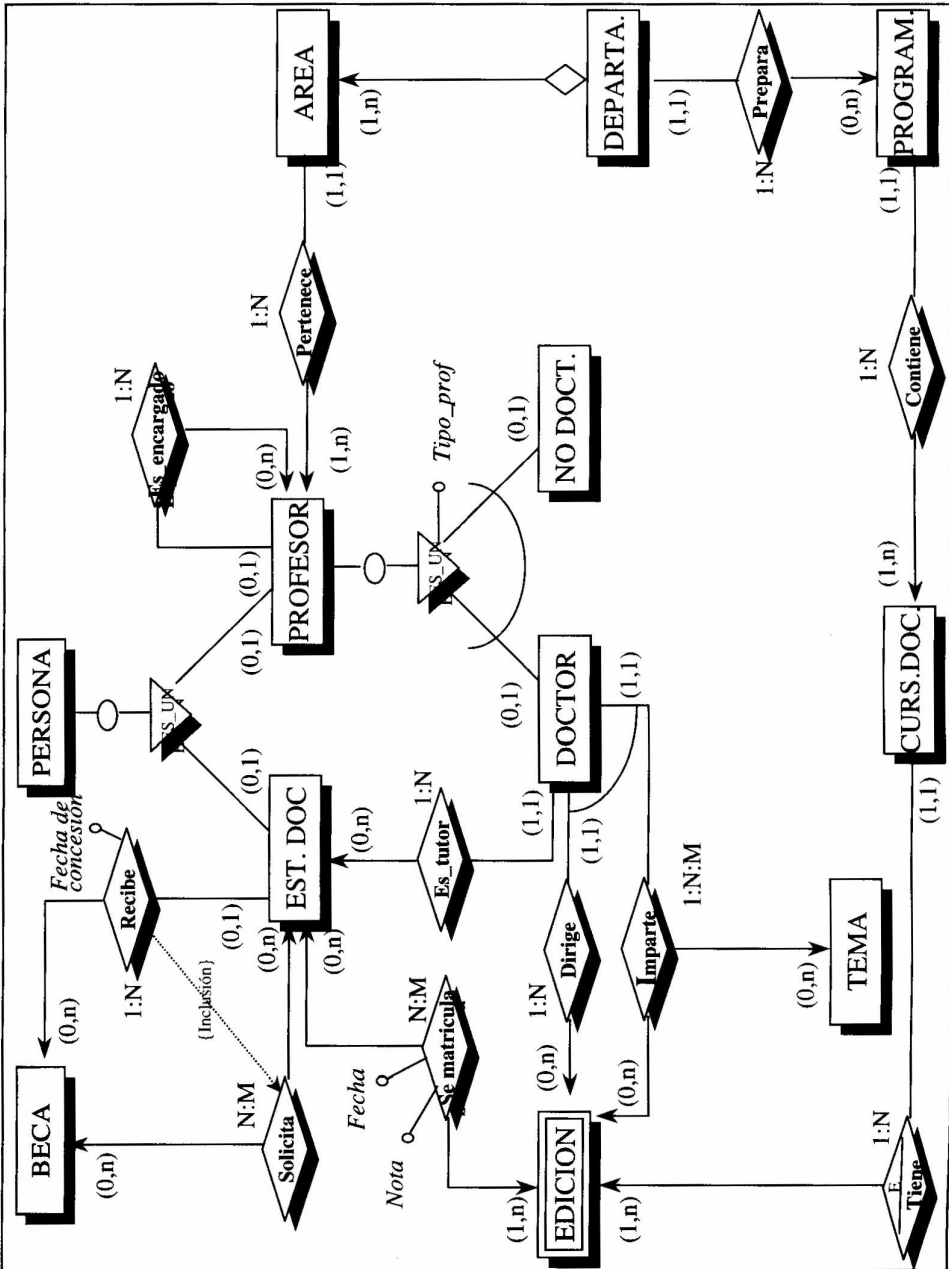
B.1. ENUNCIADO

En la Universidad Carlos III de Madrid se quiere implantar una base de datos para gestionar los cursos de doctorado que se imparten en dicha Universidad. Las especificaciones que nos han dado son las siguientes:

- Existen cursos de doctorado, de los cuales se quiere guardar información sobre su nombre, edición, fechas de inicio y fin y temario, que puede variar de una edición a otra. Se quiere saber el aula y el horario en que se imparte cada edición.
- También se quiere mantener información relacionada con las personas que participan en los cursos, que pueden ser estudiantes de doctorado o profesores. A su vez, los profesores pueden ser doctores o no doctores. Un profesor puede estar realizando un curso de doctorado, sea o no sea doctor en otra especialidad.
- Todo estudiante de doctorado tiene asignado un tutor; éste tiene que ser un doctor. Un doctor puede ser tutor de ninguno o de varios estudiantes. Un doctor no puede ser tutor de sí mismo.

- Un doctor puede dirigir o impartir ediciones de cursos, pero nunca ambas cosas. Cada edición la dirige un doctor y la pueden impartir varios; en este caso se desea conocer qué tema imparte cada uno de ellos, sabiendo que un tema, en una misma edición, sólo puede ser impartido por un doctor y que un doctor puede impartir varios temas en una misma edición.
- Un estudiante de doctorado se matricula en ediciones de cursos, deseando guardar la fecha en que se matriculó. Cuando termine el curso se desea saber la nota obtenida, mediante la cual se sabrá si el alumno aprobó o no.
- Los estudiantes pueden solicitar varias becas; se desea guardar un histórico con todas las becas concedidas a un mismo alumno y la fecha de concesión. Si no existen alumnos con los requisitos solicitados para la beca, ésta no se le concederá a nadie. De las becas se quiere saber el nombre, la edición, la cuantía y los requisitos que necesitan los solicitantes. Solo puede recibir una beca un estudiante que la ha solicitado.
- Cada departamento de la Universidad se compone de varias áreas asociadas a las cuales pertenecen los profesores. Es el departamento quien prepara los programas de doctorado; cada programa contiene varios cursos de doctorado.
- En la Universidad, todos los profesores, excepto los catedráticos, tienen un profesor responsable.

B.2. ESQUEMA CONCEPTUAL



B.3. CONSIDERACIONES SEMÁNTICAS QUE NO SE REFLEJAN EN EL ESQUEMA CONCEPTUAL

En el esquema conceptual propuesto no se reflejan algunas consideraciones semánticas que habrán de tenerse en cuenta en el correspondiente esquema lógico. En concreto no se reflejan:

1. La restricción de que un doctor no puede ser tutor de sí mismo.
2. El hecho de que, excepto los catedráticos, todos los profesores tengan un profesor responsable.

La restricción 1 no puede recogerse de ningún modo en el esquema conceptual, ya que se trata de una restricción relativa a ejemplares concretos de las entidades EST. DOCT y DOCTOR, que no pueden interrelacionarse.

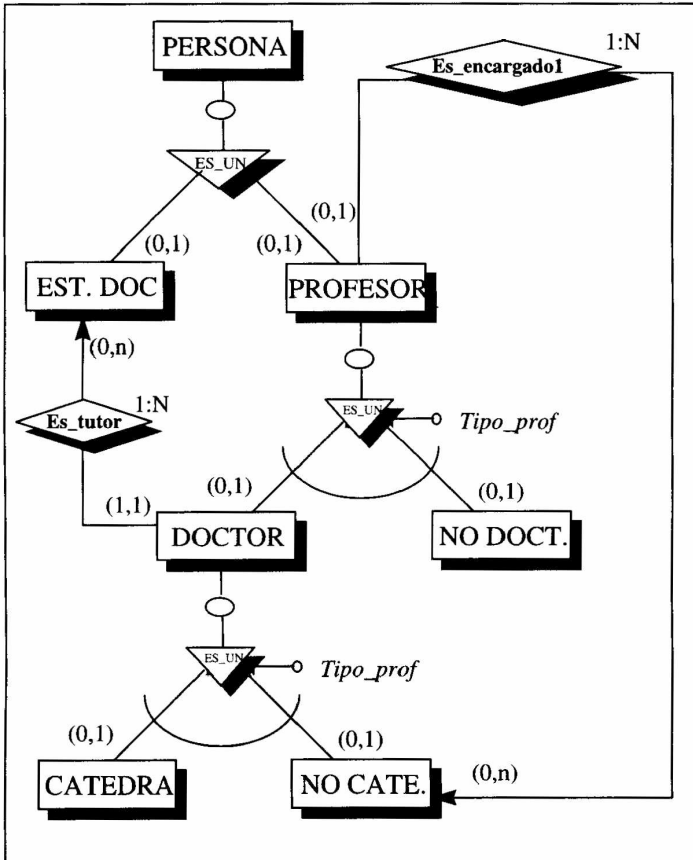


Figura B.1. Especialización de doctores en catedráticos y no catedráticos

La restricción 2 podría recogerse modificando la jerarquía de profesores del siguiente modo:

- Especializar a los doctores en catedráticos y no catedráticos, véase figura B.1.

Sin embargo, esta solución complica excesivamente el esquema en relación con la semántica que aporta, por lo que decidimos perder esta semántica y recogerla posteriormente en el esquema lógico.

B.4. DISEÑO LÓGICO ESTÁNDAR EN SQL-92

/*

Se define un dominio para cada código y para cada nombre del esquema, ya que, aunque su tipo de datos coincida, representan información distinta y por tanto no equiparable

*/

```
Create Domain C_PER Char(5);
Create Domain C_BECA Char(5);
Create Domain C_ÁREA Char(5);
Create Domain C_CUR Char(5);
Create Domain C_DEP Char(5);
Create Domain C_TEMA Char(5);
Create Domain C_PRO Char(5);
Create Domain C_EDIC Char(5);
```

```
Create Domain N_PER Char(15);
Create Domain N_BECA Char(15);
Create Domain N_ÁREA Char(25);
Create Domain N_CUR Char(25);
Create Domain N_DEP Char(25);
Create Domain N_TEMA Char(25);
Create Domain N_PRO Char(15);
```

```
Create Table DEPARTAMENTO
(Cód_dep      C_DEP      PRIMARY KEY,
 Nombre_dep   N_DEP      NOT NULL);
```

```
Create Table AREA
(Cód_area     C_ÁREA     PRIMARY KEY,
 Nombre       N_ÁREA     NOT NULL,
 Cód_dep      C_DEP,     NOT NULL,
```

```

FOREIGN KEY (Cód_dep) REFERENCES DEPARTAMENTO
                                ON UPDATE CASCADE);

Create Table PROFESOR
(Cód_profe      C_PER      PRIMARY KEY,
Nombre         N_PER      NOT NULL,
DNI            Int        NOT NULL,
Dirección      Char(25)   NOT NULL,
Materia        Char(10)
Tipo           Char       NOT NULL,
Cód_profe_e    C_PER,
Cód_area       C_ÁREA     NOT NULL,
Catedrático    Bit
FOREIGN KEY (Cód_profe_e) REFERENCES PROFESOR
                                ON UPDATE CASCADE,
FOREIGN KEY (Cód_área) REFERENCES AREA
                                ON UPDATE CASCADE,
CHECK (((Cód_prof_e = NULL) And (Catedrático= 1)) OR
      ((Cód_prof_e = NOT NULL) And (Catedrático = 0))
      /* todos los profesores, excepto los catedráticos,
        tienen un profesor responsable */
CHECK (Cód_profe <> Cód_profe_e)
      /* ningún profesor es encargado de sí mismo */
);

Create Table DOCTOR
(Cód_doctor     C_PER      PRIMARY KEY,
Anio_doc        Dec(2)    NOT NULL,
Área_doc        Char(20)  NOT NULL,
FOREIGN KEY (Cód_doctor) REFERENCES PROFESOR
                                ON DELETE CASCADE
                                ON UPDATE CASCADE);

Create Table EST_DOC
(Cód_estu       C_PER      PRIMARY KEY,
Nombre         N_PER      NOT NULL,
Apellidos      Char(30)   NOT NULL,
DNI            Int        NOT NULL,
Dirección      Char(25),
Cód_doctor     C_PER      NOT NULL,
FOREIGN KEY (Cód_doctor) REFERENCES DOCTOR
                                ON UPDATE CASCADE,
CHECK (Cód_estu <> Cód_doc));
      /* ningun doctor puede ser tutor de sí mismo*/

```


Create Table BECA

```
(Cód_beca      C_BECA    PRIMARY KEY,
 Nombre       N_BECA    NOT NULL,
 Edición      Dec(2)    NOT NULL,
 Requisitos   Char(200),
 Cuantía      Int       NOT NULL,
 Cód_estu     C_PER,
 FOREIGN KEY (Cód_estu) REFERENCES EST_DOC
                        ON DELETE SET NULL
                        ON UPDATE CASCADE);
```

Create Table SOLICITA

```
(Cód_beca      C_BECA,
 Cód_estu     C_PER,
 PRIMARY KEY (Cód_beca, Cód_estu),
 FOREIGN KEY (Cód_beca) REFERENCES BECA
                        ON DELETE CASCADE
                        ON UPDATE CASCADE,
 FOREIGN KEY (Cód_estu) REFERENCES EST_DOC
                        ON DELETE CASCADE
                        ON UPDATE CASCADE);
```

Create Assertion SOLIC_BECA

```
CHECK (( SELECT Cód_estu
          FROM Beca)
        IN (SELECT Cód_estu
            FROM Solicita
            WHERE Beca.Cód_beca=Solicita.Cód_beca));
/* sólo puede recibir una beca un alumno que la haya solicitado */
```

Create Table PROGRAMA

```
(Cód_prog      C_PRO     PRIMARY KEY,
 Nombre       N_PRO     NOT NULL,
 Cód_dep      C_DEP,
 FOREIGN KEY (Cód_dep) REFERENCES DEPARTAMENTO
                        ON UPDATE CASCADE);
```

Create Table CURSO_DOC

```
(Cód_curso     C_CUR     PRIMARY KEY,
 Nombre       N_CUR     NOT NULL,
 Núm_horas    Numeric(3),
 Cód_prog     C_PRO     NOT NULL,
 FOREIGN KEY (Cód_prog) REFERENCES PROGRAMA
                        ON DELETE CASCADE
                        ON UPDATE CASCADE);
```

Create Table TEMA

```
(Cód_tema      C_TEMA    PRIMARY KEY,
 Nombre        N_TEMA    NOT NULL,
 Materias      Char(100));
```

Create Table EDICION

```
(Cód_edición   C_EDIC,
 Cód_curso     C_CUR,
 Fecha_ini     Date,
 Fecha_fin     Date,
 Cód_doct      C_PER
 PRIMARY KEY (Cód_edición, Cód_curso),
 FOREIGN KEY (Cód_curso) REFERENCES CURSO_DOC
                        ON DELETE CASCADE
                        ON UPDATE CASCADE);
 FOREIGN KEY (Cód_doct) REFERENCES DOCTOR;
```

Create Table IMPARTE

```
(Cód_edición   C_EDIC,
 Cód_curso     C_CUR,
 Cód_tema      C_TEMA,
 Cód_doctor    C_PER    NOT NULL,
 PRIMARY KEY (Cód_edición, Cód_Curso, Cód_tema),
 FOREIGN KEY(Cód_edición,Cód_curso) REFERENCES EDICION
                        ON DELETE CASCADE
                        ON UPDATE CASCADE,
 FOREIGN KEY (Cód_tema) REFERENCES TEMA
                        ON DELETE CASCADE
                        ON UPDATE CASCADE,
 FOREIGN KEY (Cód_doctor) REFERENCES DOCTOR
                        ON DELETE CASCADE
                        ON UPDATE CASCADE);
```

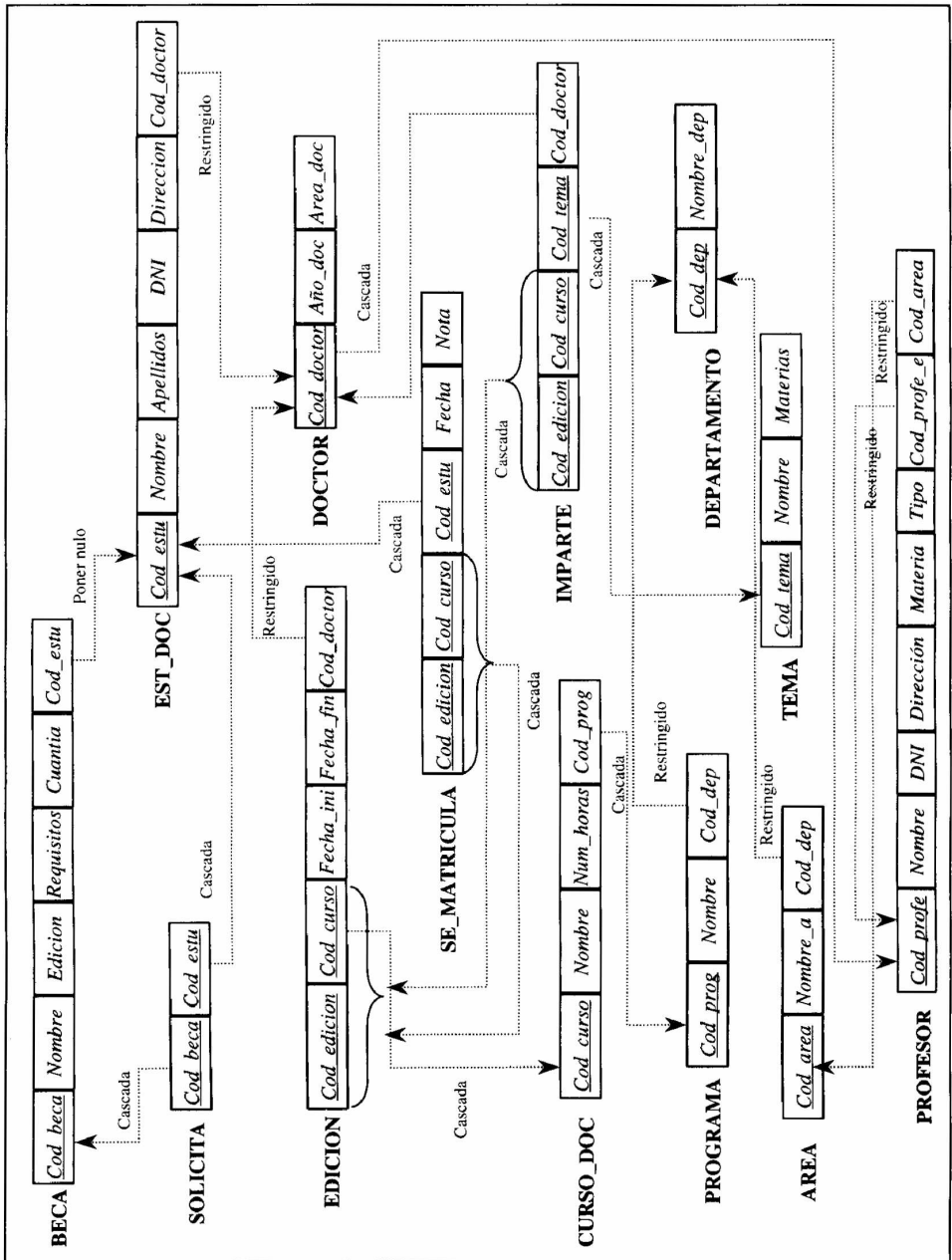
Create Assertion EDICION_IMPARTE

```
CHECK (( SELECT Cód_doctor
          FROM Edicion) NOT IN
        (SELECT Cód_doctor
          FROM Imparte));
/* un doctor imparte o bien dirige cursos */
```

Create Table SE_MATRICULA

```
(Cód_edicion   C_EDIC,
 Cód_curso     C_CUR,
 Cód_estu      C_PER,
 Fecha         Date    NOT NULL,
```

```
Nota          Char(2),
PRIMARY KEY (Cód_edicion, Cód_curso, Cód_estu),
FOREIGN KEY (Cód_edición, Cód_curso) REFERENCES EDICION,
                ON DELETE CASCADE
                ON UPDATE CASCADE,
FOREIGN KEY (Cód_estu) REFERENCES EST_DOC
                ON DELETE CASCADE
                ON UPDATE CASCADE);
```



B.6. DISEÑO LÓGICO ESPECÍFICO

B.6.1. SQLbase V4.0

/*

El diseño lógico específico se ha realizado para la versión 4.0 del SQL_BASE. Dicho gestor no soporta dominios, restricciones, ni aserciones, por lo que la semántica que llevan asociada se pierde de la definición en SQL-92 a la definición para el SQL-BASE */

```
CREATE TABLE DEPARTAMENTO
```

```
(Cód_dep      VarChar(5)  NOT NULL,
 Nombre_dep   VarChar(25) NOT NULL,
 PRIMARY KEY (Cód_dep));
```

```
CREATE UNIQUE INDEX INDDEP ON DEPARTAMENTO (Cód_dep);
```

/*

SQL-BASE soporta la sintaxis de la clave primaria pero no su semántica, por lo que obliga a definir la clave primaria como NOT NULL, así como a crear un índice único por cada clave primaria a fin de garantizar la unicidad de ésta

*/

```
CREATE TABLE AREA
```

```
(Cód_area     VarChar(5)  NOT NULL,
 Nombre       VarChar(25) NOT NULL,
 Cód_dep      VarChar(5)  NOT NULL,
 PRIMARY KEY (Cód_Area),
 FOREIGN KEY (Cód_dep) REFERENCES DEPARTAMENTO);
```

/*

SQL-BASE no soporta la definición de opciones de actualización (ON UPDATE del SQL-92). El borrado (ON DELETE), por defecto, es restringido (RESTRICT)

*/

);

```
CREATE UNIQUE INDEX INDAREA ON AREA(Cód_área);
```

```
CREATE TABLE PROFESOR
```

```
(Cód_prof     VarChar(5)  NOT NULL,
 Nombre       VarChar(15) NOT NULL,
 DNI          INT         NOT NULL,
 Dirección    VarChar(25) NOT NULL,
```

```

Materia      VarChar(10),
Tipo         Char(1)      NOT NULL,
Cód_p_e      VarChar(5),
Cód_area     VarChar(5),
             PRIMARY KEY (Cód_prof),
             FOREIGN KEY (Cód_area) REFERENCES AREA);

```

```

CREATE UNIQUE INDEX INDPROF ON PROFESOR (Cód_prof);

```

```

/*

```

Cód_prof no es único hasta que no se define un índice único. Por éste motivo, el sistema no permite definir una clave ajena sobre la tabla profesor hasta que no se crea dicho índice único

```

*/

```

```

ALTER TABLE PROFESOR
    FOREIGN KEY (Cód_p_e) REFERENCES PROFESOR
        ON DELETE CASCADE;

```

```

CREATE TABLE DOCTOR
(Cód_doct      Varchar(5)      NOT NULL,
Anio_doc       Dec(2)          NOT NULL,
Área_doc       VarChar(20)     NOT NULL,
PRIMARY KEY (Cód_doct),
FOREIGN KEY (Cód_doct) REFERENCES PROFESOR
    ON DELETE CASCADE);

```

```

CREATE UNIQUE INDEX INDDOCTOR ON DOCTOR(Cód_doct);

```

```

CREATE TABLE EST_DOC
(Cód_estu      VarChar(5)      NOT NULL,
Nombre        VarChar(15)     NOT NULL,
Apellidos     VarChar(25)     NOT NULL,
DNI           INT             NOT NULL,
Dirección     VarChar(25),
Cód_doct      VarChar(5),
PRIMARY KEY (Cód_estu),
FOREIGN KEY (Cód_doct) REFERENCES DOCTOR);

```

```

CREATE UNIQUE INDEX INDESTU ON EST_DOC(Cód_estu);

```

```

CREATE TABLE BECA
(Cód_beca      VarChar(5)      NOT NULL,
Nombre        VarChar(15)     NOT NULL,
Edición       Dec(2)          NOT NULL,
Requisitos    VarChar(200),

```

```
Cuantía          INT          NOT NULL,
Cód_Estu         VarChar(5),
PRIMARY KEY (Cód_beca),
FOREIGN KEY (Cód_estu) REFERENCES EST_DOC
                        ON DELETE SET NULL);

CREATE UNIQUE INDEX INDBECA ON BECA (Cód_beca);

CREATE TABLE SOLICITA
(Cód_beca         VarChar(5) NOT NULL,
Cód_estu         VarChar(5) NOT NULL,
PRIMARY KEY (Cód_beca,Cód_estu),
FOREIGN KEY (Cód_beca) REFERENCES BECA
                        ON DELETE CASCADE,
FOREIGN KEY (Cód_estu) REFERENCES EST_DOC
                        ON DELETE CASCADE);

CREATE UNIQUE INDEX INDSOLIC ON SOLICITA(Cód_beca,
Cód_estu);

DROP TABLE PROGRAMA;

CREATE TABLE PROGRAMA
(Cód_prog         VarChar(5) NOT NULL,
Nombre           VarChar(15) NOT NULL,
Cód_dep          VarChar(5),
PRIMARY KEY (Cód_prog),
FOREIGN KEY (Cód_dep) REFERENCES DEPARTAMENTO);

CREATE UNIQUE INDEX INDPROG ON PROGRAMA (Cód_prog);

CREATE TABLE CURSO_DOC
(Cód_curs         VarChar(5) NOT NULL,
Nombre           VarChar(25) NOT NULL,
Núm_horas        Dec(3),
Cód_prog         VarChar(5) NOT NULL,
PRIMARY KEY (Cód_curs),
FOREIGN KEY (Cód_prog) REFERENCES PROGRAMA
                        ON DELETE CASCADE);

CREATE UNIQUE INDEX INDCURSO ON CURSO_DOC (Cód_curs);

CREATE TABLE TEMA
(Cód_tema         VarChar(5) NOT NULL,
Nombre           VarChar(25) NOT NULL,
```

```
Materias      VarChar(100) NOT NULL,  
PRIMARY KEY (Cód_tema));
```

```
CREATE UNIQUE INDEX INDTEMA ON TEMA (Cód_tema);
```

```
CREATE TABLE EDICION  
(Cód_edic      VarChar(5)      NOT NULL,  
Cód_curs      VarChar(5)      NOT NULL,  
Fecha_ini     Date,  
Fecha_fin     Date,  
Cód_doct      VarChar(5),  
PRIMARY KEY (Cód_edic, Cód_curs),  
FOREIGN KEY (Cód_curs) REFERENCES CURSO_DOC  
ON DELETE CASCADE,  
FOREIGN KEY (Cód_doct) REFERENCES DOCTOR);
```

```
CREATE UNIQUE INDEX INDEDIC ON EDICION (Cód_edic,  
Cód_curs);
```

```
CREATE TABLE IMPARTE  
(Cód_edic      VarChar(5)      NOT NULL,  
Cód_curs      VarChar(5)      NOT NULL,  
Cód_tema      VarChar(5)      NOT NULL,  
Cód_doct      VarChar(5)      NOT NULL,  
PRIMARY KEY (Cód_edic, Cód_curs, Cód_tema),  
FOREIGN KEY (Cód_edic, Cód_curs) REFERENCES EDICION  
ON DELETE CASCADE,  
FOREIGN KEY (Cód_tema) REFERENCES TEMA  
ON DELETE CASCADE,  
FOREIGN KEY (Cód_doct) REFERENCES DOCTOR  
ON DELETE CASCADE);
```

```
CREATE UNIQUE INDEX INDIMPAR ON IMPARTE (Cód_edic,  
Cód_curs, Cód_tema);
```

```
CREATE TABLE SE_MATRICULA  
(Cód_edic      VarChar(5)      NOT NULL,  
Cód_curs      VarChar(5)      NOT NULL,  
Cód_estu      VarChar(5)      NOT NULL,  
Fecha         Date             NOT NULL,  
Nota          Char(2),  
PRIMARY KEY (Cód_edic, Cód_curs, Cód_estu),
```



```

FOREIGN KEY (Cód_edic, Cód_curs) REFERENCES EDICION
                                ON DELETE CASCADE,
FOREIGN KEY (Cód_estu) REFERENCES EST_DOC
                                ON DELETE CASCADE);

CREATE UNIQUE INDEX INDMATRIC ON SE_MATRICULA
(Cód_edic, Cód_curs, Cód_estu);

```

B.6.2. Oracle v8.0.5

```
/*
```

En este otro ejemplo, la implementación se ha realizado para Oracle versión 8.0.5. Oracle no admite los dominios y la única cláusula que admite en las claves ajenas es ON DELETE CASCADE. También en este caso, como en el ejemplo para SQLBase, se pierde la semántica que llevan asociadas las demás restricciones. En cuanto a las aserciones, Oracle permite implementarlas como disparadores de base de datos, como se muestra en este ejemplo con la cláusula CREATE TRIGGER.

```
*/
```

```

Create Table DEPARTAMENTO
(Cód_dep      Varchar2(5)    PRIMARY KEY,
Nombre_dep   Varchar2(25)   NOT NULL);

```

```

Create Table AREA
(Cód_área     Varchar2(5)    PRIMARY KEY,
Nombre       Varchar2(25)   NOT NULL,
Cód_dep      Varchar2(5),   NOT NULL,
FOREIGN KEY (Cód_dep)
REFERENCES DEPARTAMENTO(Cód_dep)
);

```

```

Create Table PROFESOR
(Cód_profe    Varchar2(5)    PRIMARY KEY,
Nombre       Varchar2(15)   NOT NULL,
DNI          Number        NOT NULL,
Dirección    Varchar2(25)   NOT NULL,
Materia      Varchar2(10)
Tipo         Char          NOT NULL,
Cód_profe_e  Varchar2(5),
Cód_área     Varchar2(5)    NOT NULL,
Catedrático  Char
FOREIGN KEY (Cód_profe_e)
REFERENCES PROFESOR(Cód_profe),
FOREIGN KEY (Cód_área) REFERENCES AREA(Cód_area),

```

```

CHECK (((Cód_prof_e IS NULL) And (Catedrático= 1)) OR
      ((Cód_prof_e IS NOT NULL) And (Catedrático= 0))
      /* todos los profesores, excepto los catedráticos,
         tienen un profesor responsable */
CHECK (Cód_profe <> Cód_profe_e)
      /* ningún profesor es encargado de sí mismo */
);

```

```

Create Table DOCTOR
(Cód_doctor      VARCHAR2(5)      PRIMARY KEY,
Anio_doc         Number(2)        NOT NULL,
Área_doc         Varchar2(20)     NOT NULL,
FOREIGN KEY (Cód_doctor)
      REFERENCES PROFESOR(Cód_profe)
      ON DELETE CASCADE
);

```

```

Create Table EST_DOC
(Cód_estu        VARCHAR2(5)      PRIMARY KEY,
Nombre           Varchar2(15)     NOT NULL,
Apellidos        Varchar2(30)     NOT NULL,
DNI              Number            NOT NULL,
Dirección        Varchar2(25),
Cód_doctor       Varchar2(5)      NOT NULL,
FOREIGN KEY (Cód_doctor) REFERENCES DOCTOR,
CHECK (Cód_estu <> Cód_doc));
      /* ningún doctor puede ser tutor de sí mismo*/

```

```

Create Table BECA
(Cód_beca        Varchar2(5)      PRIMARY KEY,
Nombre           Varchar2(15)     NOT NULL,
Edición          Number(2)        NOT NULL,
Requisitos       Varchar2(200),
Cuantía          Number            NOT NULL,
Cód_estu         Varchar2(5),
FOREIGN KEY (Cód_estu) REFERENCES EST_DOC
);

```

```

Create Table SOLICITA
(Cód_beca        Varchar2(5)      NOT NULL,
Cód_estu         Varchar2(5)      NOT NULL,
PRIMARY KEY (Cód_beca, Cód_estu),
FOREIGN KEY (Cód_beca) REFERENCES BECA(Cód_beca)
      ON DELETE CASCADE,

```

```
FOREIGN KEY (Cód_estu) REFERENCES EST_DOC(Cód_estu)
ON DELETE CASCADE);
```

```
Create Trigger SOLIC_BECA After Insert Or Update
Of Cód_estu On BECA For Each Row
```

```
Begin
```

```
  Select Cód_estu
```

```
    From Solicita where Cód_estu = :new.Cód_estu;
```

```
Exception
```

```
  When No_Data_Found then
```

```
    Raise_Application_Error(-20000,'El estudiante no
ha solicitado la beca');
```

```
End;
```

```
/* sólo puede recibir una beca un alumno que la haya solicitado */
```

```
Create Table PROGRAMA
```

```
(Cód_prog      Varchar2(5)    PRIMARY KEY,
```

```
  Nombre      Varchar2(15)  NOT NULL,
```

```
  Cód_dep     Varchar2(25),
```

```
FOREIGN KEY (Cód_dep)
```

```
  REFERENCES DEPARTAMENTO(Cód_dep));
```

```
Create Table CURSO_DOC
```

```
(Cód_curso     Varchar2(5)    PRIMARY KEY,
```

```
  Nombre      Varchar2(25)  NOT NULL,
```

```
  Núm_horas   Number(3),
```

```
  Cód_prog    Varchar2(5)    NOT NULL,
```

```
FOREIGN KEY (Cód_prog) REFERENCES PROGRAMA(Cód_prog)
ON DELETE CASCADE);
```

```
Create Table TEMA
```

```
(Cód_tema      Varchar2(5)    PRIMARY KEY,
```

```
  Nombre      Varchar2(25)  NOT NULL,
```

```
  Materias    Varchar2(100));
```

```
Create Table EDICION
```

```
(Cód_edición   Varchar2(5),
```

```
  Cód_curso    Varchar2(5),
```

```
  Fecha_ini    Date,
```

```
  Fecha_fin    Date,
```

```
  Cód_doct     Varchar2(5)
```

```
PRIMARY KEY (Cód_edicion, Cód_curso),
```

```
FOREIGN KEY (Cód_curso)
```

```
REFERENCES CURSO_DOC(Cód_curso)ON DELETE CASCADE),  
FOREIGN KEY(Cód_doct) REFERENCES DOCTOR(Cód_doctor));
```

```
Create Table IMPARTE  
(Cód_edición      Varchar2(5) NOT NULL,  
  Cód_curso       Varchar2(5) NOT NULL,  
  Cód_tema        Varchar2(5) NOT NULL,  
  Cód_doctor      Varchar2(5) NOT NULL,  
PRIMARY KEY (Cód_edicion, Cód_Curso, Cód_tema),  
FOREIGN KEY(Cód_edicion,Cód_curso)  
  REFERENCES EDICION(Cód_edición, Cód_curso)  
    ON DELETE CASCADE,  
FOREIGN KEY (Cód_tema) REFERENCES TEMA(Cód_tema)  
    ON DELETE CASCADE,  
FOREIGN KEY(Cód_doctor) REFERENCES DOCTOR(Cód_doctor)  
    ON DELETE CASCADE);
```

```
Create Trigger EDICION_IMPARTE After Insert Or Update  
Of Cód_doctor On IMPARTE For Each Row
```

```
Begin
```

```
  Select Cód_doctor
```

```
    From Edicion where Cód_doctor = :new.Cód_doctor;
```

```
  Raise_Application_Error(-20001,'El Doctor ya  
    dirige un curso');
```

```
Exception
```

```
  When Others then
```

```
    Null;
```

```
End;
```

```
Create Trigger IMPARTE_EDICION After Insert Or Update  
Of Cód_doctor On EDICION For Each Row
```

```
Begin
```

```
  Select Cód_doctor
```

```
    From Imparte where Cód_doctor = :new.Cód_doctor;
```

```
  Raise_Application_Error(-20002,'El Doctor ya  
    imparte un curso');
```

```
Exception
```

```
  When Others then
```

```
    Null;
```

```
End;
```

/ un doctor o imparte o dirige cursos */*

```
Create Table SE_MATRICULA
(Cód_edición    Varchar2(5) NOT NULL,
 Cód_curso      Varchar2(5),
 Cód_estu       Varchar2(5),
 Fecha          Date      NOT NULL,
 Nota           Varchar2(2),
PRIMARY KEY (Cód_edición, Cód_curso, Cód_estu),
FOREIGN KEY(Cód_edición,Cód_curso)
REFERENCES EDICION(Cód_edición,Cód_curso),
                ON DELETE CASCADE,
FOREIGN KEY (Cód_estu) REFERENCES EST_DOC(Cód_estu)
                ON DELETE CASCADE);
```

Diseño de Bases de Datos Relacionales

La creación de un Sistema de Información abarca dos grandes áreas claramente diferenciadas: los datos y los tratamientos, que aunque estén muy relacionadas y su concepción y diseño no puedan realizarse de forma independiente, sus problemas a resolver son de naturaleza distinta. Sin embargo, y a pesar de sus más de tres décadas de existencia, de sus miles de usuarios en el mundo entero, y de la extraordinaria atención que han dedicado al tema científicos y técnicos de reconocida valía, la concepción y diseño de una base de datos sigue resultando una tarea larga, difícil y costosa que no debe improvisarse, ya que lleva consigo una serie de actividades de decisión y planificación muy complejas y variadas. Estas dificultades inherentes al diseño de una base de datos han de tener una adecuada respuesta metodológica.

El objetivo principal de este libro es, precisamente, proporcionar unos principios metodológicos que ayuden a realizar un buen diseño conceptual (utilizando el modelo E/R extendido) y a llevar a cabo la transformación del esquema conceptual obtenido a un esquema lógico con la mínima pérdida de semántica. También se suministra una sólida base teórica, como es la teoría de la normalización, al diseño lógico de bases de datos. Finalmente, se analiza el soporte que pueden ofrecer las herramientas CASE y los diccionarios de recursos de información al desarrollo de bases de datos.



Se incluye con el libro un disquete que contiene la herramienta RENO (RElaciones NOrmalizadas), en la que se encuentra un ejecutable que implementa diversos algoritmos de normalización, así como varios archivos con dependencias que sirven como ejemplos de prueba de los algoritmos. RENO tiene una interfaz de usuario muy fácil de usar "amigable" y una ayuda, lo que hace muy cómoda su utilización. Puede resultar interesante para profesores y alumnos que desean comprobar si sus ejercicios han sido resueltos correctamente.

Se trata de un libro dirigido tanto a alumnos de asignaturas de Bases de Datos en facultades o escuelas universitarias, como a estudiantes de ciclos formativos de Técnicos en Desarrollo de Aplicaciones Informáticas, que preferentemente estudien el desarrollo de aplicaciones de cuarta generación, con herramientas CASE. También resulta muy adecuado para profesionales de la informática que estén trabajando en áreas de bases de datos, y usuarios avanzados o directivos con responsabilidades en el desarrollo y explotación de sistemas.

Este libro conjuga aspectos teóricos y prácticos, poniendo al alcance de los lectores la experiencia de los autores en la aplicación, investigación y docencia en el área de las bases de datos.

ISBN 970-15-0526-3



9 789701 505267