Problemas de BASES de DATOS

3ª Edición

SANZ Y TORRES

Luis Grau Fernández
Ignacio López Rodriguez

PROBLEMAS de Bases de Datos

3ª Edición

PROBLEMAS de Bases de Datos

3ª Edición

PROBLEMAS de Bases de Datos

V Edición

PROBLEMAS de Bases de Datos

3ª Edición

Luis Grau Fernández

Ignacio López Rodríguez

SANZ Y TORRES

1.ª edición: marzo 1998
2.ª edición: febrero 2001
3.ª edición: marzo 2006

PROBLEMAS DE BASES DE DATOS

Pinos Alta, 49 - 28029 Madrid
Teléfs.: 902 400 415 - 91 314 55 99
www.sanzytorres.com
libreria@sanzytorres.com
www.sanzytorres.com/editorial
editorial@sanzytorres.com

ISBN: 84-96094-69-3
Depósito legal: M-8.888-2006

Composición y Portada:
Autores
Impresión:
Edigrafos, S. A., c/ Volta, 2, Pol. Ind. San Marcos, 28906 Getafe (Madrid)
Encuadernación:
Felipe Méndez, S. A. c/ Del Carbón, 6 y 8, Pol. Ind. San José de Valderas 2, 28918 Leganés (Madrid)

Prólogo

Objetivos

El objetivo básico de este libro, es servir de apoyo al estudio de las asignaturas de *Bases de Datos* que se imparten en las Facultades y Escuelas de Informática.

Existen actualmente excelentes libros de texto que tratan, desde un punto de vista teórico, la mayoría de los temas que suelen incluirse en un curso de introducción a las bases de datos. No obstante, la cantidad de problemas resueltos que se pueden encontrar en estos libros es realmente escasa y por ello, de alguna manera, este texto viene a suplir ese vacío.

A quien va dirigido este libro

Problemas de Bases de Datos, está concebido como un libro de apoyo para la asignatura *Bases de datos* que tienen que cursar los alumnos de la Escuela Técnica superior de Informática de la UNED en el segundo curso de las carreras de Ingeniería Técnica en Informática de Sistemas e Ingeniería Técnica en Informática de Gestión. Este hecho impone una serie de ligaduras sobre la estructuración de sus contenidos y la forma de desarrollarlos ya que tienen que ser, en la medida de lo posible, autosuficientes y comprensibles a estudiantes que no asisten regularmente a clase.

Esta reflexión, ha llevado a los autores a incluir en cada capítulo un breve repaso sobre las herramientas y conceptos que se utilizan en la resolución de los problemas. En estas introducciones teóricas, se ha pretendido unificar las diversas notaciones existentes, ya que la mayoría de los libros de texto que tratan sobre el tema utilizan distintas nomenclaturas que, aun siendo todas ellas correctas, pueden confundir al alumno cuando éste, al estudiar la asignatura, ha tenido que consultar varios de ellos.

Debido a que en la mayoría de Escuelas y Facultades de Informática los temarios se aproximan bastante en su contenido, este libro puede resultar también útil a todos aquellos estudiantes que tengan que cursar la asignatura *Bases de Datos* en sus centros de estudios.

Organización del texto

El texto está dividido en diez capítulos cuyos contenidos son:

- *Capítulo 1. Modelo Entidad-Relación.* Se exponen los conceptos generales de diseño de bases de datos utilizando el modelo Entidad-Relación para lo cual, además de realizar un breve repaso sobre el tema, se resuelven y se comentan siete problemas de diversa dificultad. Se ha intentado de alguna manera que, a través de estos problemas, se pongan de manifiesto la mayoría de los inconvenientes que pueden surgir al realizar un diseño real.

- *Capítulo 2. Álgebra Relacional.* Su objetivo principal es conseguir que el alumno adquiera la destreza necesaria a la hora de realizar consultas sobre una base de datos utilizando el álgebra relacional. La resolución de los problemas, por parte del alumno, es de vital importancia para luego enfrentarse de una manera óptima al estudio de lenguajes relacionales como SQL.

- *Capítulo 3. Cálculo Relacional de Tuplas.* Se estudia la manera de realizar consultas sobre una base de datos utilizando el cálculo relacional de tuplas. Aunque al principio del capítulo se da una introducción teórica sobre el tema, sería aconsejable que los alumnos hiciesen un breve repaso sobre los conceptos estudiados en la asignatura *Lógica Matemática*. Esta variante del cálculo relacional es sobre la que se fundamentan lenguajes comerciales como QUEL.

- *Capítulo 4. Cálculo Relacional de Dominios.* Se aborda la realización de consultas sobre una base de datos utilizando el cálculo relacional de dominios. Al igual que en el capítulo anterior, sería aconsejable que los alumnos hiciesen un breve repaso sobre los conceptos estudiados en la asignatura *Lógica Matemática*. Esta variante del cálculo relacional es sobre la que se fundamentan lenguajes comerciales como QBE.

- *Capítulo 5. SQL.* Se resuelven problemas sobre manejo y mantenimiento de una base de datos utilizando el lenguaje SQL. Es importante que, a la hora de enfrentarse a estos problemas, el alumno haya resuelto y comprendido previamente los del capítulo 2.

- *Capítulo 6. QBE.* Se resuelven problemas sobre manejo y mantenimiento de una base de datos, utilizando el lenguaje QBE. Es importante que a la hora de abordar estos problemas, el alumno haya resuelto y comprendido previamente los del capítulo 2 y 4.

- *Capítulo 7. QUEL.* Se resuelven problemas sobre manejo y mantenimiento de una base de datos, utilizando el lenguaje QUEL. Es importante que a la hora de abordar estos problemas, el alumno haya resuelto y comprendido previamente los problemas del capítulo 2 y 3.
- *Capítulo 8. Formas Normales.* Se plantean y resuelven diez problemas, que a nuestro juicio consideramos fundamentales para poder decidir si una base de datos es formalmente correcta o no. En estos problemas se presentan la mayoría de las dificultades con las que se puede encontrar el alumno al estudiar este tema.
- *Capítulo 9. Modelo en Red.* Se presenta el modelo en Red que, a pesar de no utilizarse hoy en día con tanta frecuencia como el modelo relacional, es lo suficiente importante como para dedicarle un tema completo. Además de exponer los conceptos teóricos básicos, se resuelven una serie de problemas de manejo y mantenimiento de un sistema en red, utilizando como lenguaje anfitrión PASCAL.
- *Capítulo 10. Modelo Jerárquico.* Se presenta el modelo Jerárquico el cual fue durante una serie de años el más utilizado por grandes entidades como bancos. Hoy en día se tiende a sustituir los sistemas basados en este modelo por sistemas relacionales más modernos y potentes. No obstante, existen todavía muchos sistemas que hacen uso del modelo jerárquico a la hora de almacenar sus datos. Además de exponer los conceptos teóricos básicos, se resuelven una serie de problemas de manejo y mantenimiento de un sistema jerárquico, utilizando como lenguaje anfitrión PASCAL.

Metodología

Se ha tratado de cuidar de manera muy especial los aspectos específicos de la enseñanza a distancia. Los conceptos teóricos, se van introduciendo en los problemas de forma progresiva, de manera que el estudiante puede ir avanzando a su propio ritmo.

La estructura de todos los capítulos es uniforme. En ellos, además de los problemas resueltos, se realiza una breve introducción teórica con la finalidad de que el alumno disponga de las herramientas básicas utilizadas en la resolución de los problemas.

Agradecimientos

Los autores agradecen a sus compañeros del Departamento de Informática y Automática de la UNED su apoyo inestimable en todo momento. En particular a los profesores D. Juan Carlos Lázaro Obensa, por su esmerado diseño de la portada, y a D. José Jiménez González, por sus sugerencias en los capítulos de álgebra y cálculo relacional. También nuestro más sincero agradecimiento a los alumnos de la Escuela Universitaria de Informática de la UNED, por sus constructivas críticas. En especial a Dña. Mercedes Toral Domínguez, a D. José Mª Torralba Martínez y a D. Luis Moreno, por su cuidadosa revisión del texto.

Los autores

Índice

Capítulo 1. Modelo Entidad-Relación 1

1.1 Introducción 1
1.2 Entidades y conjunto de entidades 2
1.3 Relaciones y conjuntos de relaciones 2
1.4 Claves 3
1.5 Diagramas Entidad-Relación 3
1.6 Cardinalidad de asignación 4
1.7 Reducción de los diagramas E-R a tablas 6
1.8 Problemas resueltos 6

Capítulo 2. Álgebra Relacional 33

2.1 Introducción 33
2.2 Estructura de datos relacional 33
2.3 Álgebra relacional 37
2.3.1 Operación renombrar 37
2.3.2 Operaciones de conjuntos 38
2.3.3 Operaciones relacionales 40
2.3.4 Operadores adicionales 42
2.3.5 Asignación relacional 42
2.4 Problemas resueltos 43

Capítulo 3. Cálculo Relacional de Tuplas 83

3.1 Introducción 83
3.2 Cálculo relacional de tuplas 83
3.3 Problemas resueltos 85

Capítulo 4. Cálculo Relacional de Dominios 117

4.1 Introducción 117
4.2 Cálculo relacional de dominios 117
4.3 Problemas resueltos 119

Capítulo 5. SQL 155

5.1 Introducción 155
5.2 Formas de uso del SQL 156
5.3 Partes del SQL 156
5.4 Tipos de datos 156
5.5 Creación de tablas 157
5.6 Modificación de tablas 157
5.7 Eliminación de tablas 158
5.8 Índices 158
5.9 Comando SELECT 158
5.10 Filtros en SQL 158
5.11 La operación renombramiento 159
5.12 Ordenación de resultados 160
5.13 Funciones agregadas 160
5.14 Agrupación de filas 160
5.15 Problemas resueltos 161

Capítulo 6. Query-By-Example 191

6.1 Introducción 191
6.2 Consultas en Query-by-Example 191
6.3 Operaciones permitidas en Query-by-Example 193
6.4 Problemas resueltos 195

Capítulo 7. QUEL 225

7.1 Introducción 225
7.2 Tipos de datos 226
7.3 Definición de datos 226
7.4 Consultas en QUEL 227
7.5 Operaciones permitidas en QUEL 229
7.6 Modificación de la base de datos 230
7.7 Problemas resueltos 231

Capítulo 8. Formas Normales 267

8.1 Introducción 267
8.2 Dependencias funcionales 268
8.3 Dependencias multivaluadas 269

8.4 Dependencias de reunión 269
8.5 Primera forma normal 270
8.6 Segunda forma normal 270
8.7 Tercera forma normal 271
8.8 Forma normal de Boyce-Codd 271
8.9 Cuarta forma normal 272
8.10 Quinta forma normal 272
8.11 Problemas resueltos 273

Capítulo 9. Modelo en Red 307

9.1 Introducción 307
9.2 Registros, tipos de registros y datos 308
9.3 Conjuntos, Tipos de Conjuntos e Instancias 308
9.4 Restricciones de los miembros de un conjunto 310
9.5 Tipos especiales de conjuntos. 311
9.6 Area de trabajo de programa 312
9.7 DDL de un sistema en red 313
9.8 Base de datos de ejemplo 313
9.9 DML de un sistema en RED 314
9.10 Comandos de recuperación y navegación 315
9.11 Comandos de Actualización 318
9.12 Problemas resueltos 319

Capítulo 10. Modelo Jerárquico 329

10.1 Introducción 329
10.2 Estructura jerárquica de datos 330
10.3 Nomenclatura adicional 331
10.4 Area de trabajo de programa 332
10.5 DDL de un sistema jerárquico 333
10.6 DML de un sistema jerárquico 333
10.7 Cálculo de funciones de agregados 338
10.8 Actualización de registros 339
10.9 Problemas resueltos 339

Apéndice A. Base de Datos Automóviles 349

A.1 Introducción 349
A.2 Modelo Entidad-Relación 349
A.3 Tablas 350

Bibliografía 351

Capítulo 1
Modelo Entidad-Relación

1.1 Introducción

El *Modelo Entidad-Relación (E-R)*, también conocido como *Modelo Conceptual de Datos*, es una técnica especial de representación gráfica que incorpora información relativa a los datos y la relación existente entre ellos para proporcionar una visión del mundo real. En la actualidad, prácticamente todas las metodologías de diseño de sistemas tienen incorporado el Modelo Entidad-Relación dentro de su diseño de datos.

La realización de un Modelo E-R, supone siempre un paso previo al futuro diseño de una base de datos en cualquiera de los enfoques existentes (Relacional, Jerárquico, en Red, etc.). Por ello, esta metodología se puede utilizar como punto de partida común para posteriormente analizar el diseño de los diferentes modelos conceptuales, cada uno con su visión particular de los datos.

Las características del Modelo E-R son las siguientes:

- Refleja tan sólo la existencia de los datos, no lo que se hace con ellos.
- Se incluyen todos los datos del sistema en estudio, y por tanto no está orientado a aplicaciones particulares.
- Es independiente de las bases de datos y sistemas operativos concretos.
- No se tienen en cuenta restricciones de espacio, almacenamiento, ni tiempo de ejecución.
- Está abierto a la evolución del sistema.

Por tanto, en el Modelo E-R se da una visión del mundo real con la mayor naturalidad mediante los objetos y sus relaciones, de manera que a partir de ahí su

implementación permita mantener las propiedades de las bases de datos. El Modelo E-R se basa en la percepción de un mundo real que consiste en un conjunto de objetos básicos denominados *Entidades*, así como de las *Relaciones* entre ellos.

1.2 Entidades y conjuntos de entidades

Una *Entidad* es una cosa u objeto concreto o abstracto que existe, que puede distinguirse de otros y del cual se desea almacenar información. Por ejemplo, 'Juan Fernández' con *dni* "23.321.321" es una entidad ya que identifica únicamente una persona específica en el universo. Tal y como se ha dicho antes, una entidad puede ser concreta como en el caso anterior (una persona) o abstracta como un día festivo o un concepto.

Un *Conjunto de Entidades* es, como su nombre indica, un grupo de entidades del mismo tipo. Por ejemplo el conjunto de todos los alumnos matriculados en la universidad puede definirse como el conjunto de entidades ALUMNO.

Una entidad está siempre representada por un conjunto de *Atributos* que representan características de la misma. Por ejemplo el conjunto de entidades ALUMNO puede tener como atributos el *nombre,* la *fecha de nacimiento* y el *dni.*

Por tanto una base de datos incluirá una colección de conjuntos de entidades, cada uno de los cuales contendrá un número cualquiera de entidades del mismo tipo.

1.3 Relaciones y conjuntos de relaciones

Una *Relación* es una asociación entre varias entidades. Por ejemplo se puede definir una relación que asocie al alumno de *nombre* 'Juan' con la asignatura 'bases de datos'. Esta sería una relación que asociaría la entidad ALUMNO de *nombre* "Juan", con la entidad ASIGNATURA de *nombre* "Bases de Datos".

Un *Conjunto de Relaciones* es un grupo de relaciones del mismo tipo. Por ejemplo, supónganse los dos conjuntos de entidades ALUMNO y ASIGNATURA. Se puede entonces definir el conjunto de relaciones ALUM-ASIG para denotar la asociación entre los alumnos y las asignaturas en las que están matriculados. Este es un ejemplo de relación binaria, es decir, entre dos entidades, aunque en general, los conjuntos de relaciones pueden ser n-arias es decir, entre n conjuntos de entidades distintos.

Una relación también puede tener *Atributos de Relación* los cuales representarán características propias de la asociación entre varias entidades. Por ejemplo, la relación anteriormente definida ALUM-ASIG podría tener definido un

atributo llamado *nota,* el cual expresaría la calificación obtenida por un alumno en una asignatura.

1.4 Claves

Una *Clave* es un conjunto de uno o más atributos que, considerados conjuntamente, nos permiten identificar de forma única a una entidad dentro de un conjunto de entidades. A esta definición hay que añadir que este conjunto de atributos debe de ser mínimo, en el sentido de que ningún subconjunto de atributos de la clave pueda funcionar también como clave.

Siguiendo con el ejemplo anterior, para el caso del conjunto de entidades ALUMNO el atributo *dni* puede funcionar como clave, ya que cada entidad dentro del conjunto de entidades tiene un *dni* distinto.

En un conjunto de entidades puede existir más de una clave. A todas las posibles claves existentes se las denominará *claves candidatas.* Se usará el término *clave primaria* para denotar una clave candidata que elige el diseñador de la base de datos como el medio principal de identificar entidades dentro de un conjunto de entidades.

En función de las claves, las entidades se pueden clasificar como:

- *Entidades fuertes*: Son aquellos conjuntos de entidades que tienen una clave primaria.
- *Entidades débiles*: Son aquellos conjuntos de entidades que no tienen los atributos necesarios como para definir una clave primaria y dependen de una entidad fuerte. No obstante, es necesario dotar a este tipo de entidades con algún tipo de clave. Para ello se define el concepto de *discriminador* como el conjunto de atributos X de la entidad débil tal que, para cada valor de la clave primaria Y de la entidad fuerte a la que está supeditada, el valor de X identifica de manera única una entidad del conjunto de entidades débiles. Se profundizará sobre este concepto en los problemas resueltos. Por ello la *clave primaria* de un conjunto de entidades débiles está formada por la clave primaria del conjunto de entidades fuertes y el discriminador.

Por último se definirá el concepto de *clave ajena* como aquel conjunto de atributos de una entidad que son clave primaria de otra entidad.

1.5 Diagramas Entidad-Relación

La estructura lógica global de una base de datos puede representarse gráficamente por medio de un *Diagrama Entidad-Relación* el cual consta de los siguientes elementos:

- *Rectángulos*: Representan conjuntos de entidades. Si son débiles, los rectángulos serán dobles.
- *Elipses*: Representan atributos
- *Rombos*: Representan conjuntos de relaciones
- *Líneas*: Enlazan atributos a conjuntos de entidades, atributos a conjuntos de relaciones y conjuntos de entidades a conjuntos de relaciones.

Supóngase por ejemplo que se desea realizar un modelo Entidad-Relación de la base de datos de ejemplo del apartado 1.3, es decir, de ASIGNATURAS, ALUMNOS y de las relaciones existentes entre los dos conjuntos de entidades como por ejemplo, el poder conocer a los alumnos que están matriculados en una determinada asignatura y la nota obtenida.

Un modelo Entidad-Relación que contenga estas especificaciones será el siguiente:

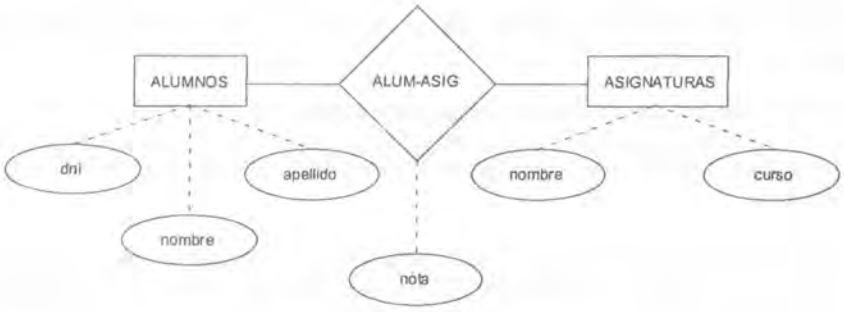


Figura 1.1: Ejemplo de modelo E-R

Como puede verse, existen dos conjuntos de entidades, ALUMNOS y ASIGNATURAS, que representan los dos objetos sobre los cuales se desea guardar información. Además se ha añadido una relación denominada ALUM-ASIG que representa la asociación entre un alumno y una asignatura. Esta relación presenta un atributo propio denominado *nota*, que representa la calificación obtenida por un alumno en una asignatura.

1.6 Cardinalidad de asignación

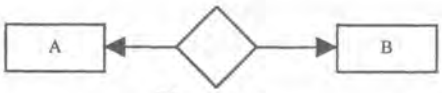
La *cardinalidad de asignación* es una restricción que expresa el número de entidades con las que puede asociarse otra entidad, mediante un conjunto de relaciones. Las cardinalidades de asignación son más útiles para

describir conjuntos binarios de relaciones, aunque ocasionalmente contribuyen a la descripción de conjuntos de relaciones que implican más de dos conjuntos de entidades.

Para un conjunto binario de relaciones R entre los conjuntos de entidades A y B, la cardinalidad de asignación debe ser una de las siguientes:

- *Una a una*. Una entidad A está asociada a lo sumo con una entidad B y viceversa.
- *Una a muchas*. Una entidad A está asociada con un número cualquiera de entidades B, sin embargo una entidad B puede estar asociada a lo sumo con una entidad A.
- *Muchas a una*. Una entidad A está asociada a lo sumo con una entidad B, aunque una entidad B puede estar asociada a un número cualquiera de entidades A.
- *Muchas a muchas*. Una entidad A está asociada con un número cualquiera de entidades B, y viceversa.

La cardinalidades de asignación se indican en los modelos E-R mediante el uso de flechas.



Una a una

Una a muchas

Muchas a una

Muchas a muchas

1.7 Reducción de los diagramas E-R a tablas

Todo modelo E-R puede representarse por medio de tablas relacionales. Para ello las reglas son las siguientes:

- Para cada conjunto de entidades fuertes A, existe una única tabla a la que se le asigna el nombre del conjunto de entidades A, y cuyos atributos son los atributos del conjunto de entidades.
- Para cada conjunto de entidades débiles B, existe una única tabla a la que se le asigna el nombre de la entidad débil B, y cuyos atributos son los atributos de la entidad débil más los de la clave primaria de la entidad fuerte a la que está subordinada.
- Para cada conjunto de relaciones existe una única tabla a la que se le asigna el nombre del conjunto de relaciones, y cuyos atributos son las claves primarias de todas las entidades que relaciona más los atributos propios de la relación.

1.8 Problemas resueltos

♦ Problema 1.1

Se desea diseñar un esquema relacional de una base de datos para un centro de enseñanza que contenga información sobre los alumnos, las asignaturas y las calificaciones que se obtienen en cada una de las mismas. Desarrollar un modelo E-R del mismo y posteriormente reducirlo a tablas.

Solución:

Como puede deducirse del enunciado anterior, existen dos entidades que se nombrarán como ASIGNATURAS y ALUMNOS cada una de ellas con sus atributos propios, es decir, para ALUMNOS (nombre, apellido, dni, domicilio) y para ASIGNATURAS (nomasig, curso).

Además hay que tener en cuenta que, según se especifica en el enunciado, debe existir información sobre las calificaciones que ha obtenido cada alumno en cada asignatura, por lo que se hace necesario el uso de una relación entre ambas entidades que se nombrará como ALUM-ASIG con el atributo propio *nota*.

Visto esto, el modelo E-R resultante debe estar compuesto por dos entidades, ASIGNATURAS y ALUMNOS, y una relación entre ambas con el atributo propio NOTAS.

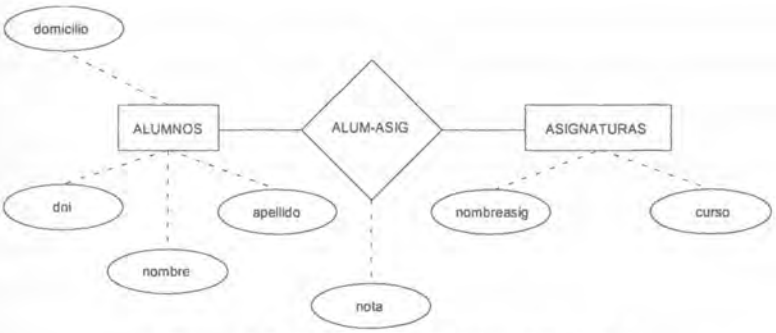


Figura 1.2: Diagrama E-R del problema 1.1

Es importante darse cuenta de que como un alumno puede estar matriculado en muchas asignaturas, y en una asignatura pueden estar matriculados muchos alumnos, la asignación de cardinalidad que se ha hecho es de muchos a muchos.

Antes de continuar es necesario definir, si es que existen, cuáles serán las claves primarias de cada una de las entidades. Para ello se subrayará el conjunto de atributos que puedan desempeñar dicha función en cada una de la entidades.

ALUMNOS (dni, nombre, apellido, domicilio)
ASIGNATURAS (nombreasig, curso)

El siguiente paso, consiste en pasar este diseño a un modelo relacional para lo cual basta con aplicar las tres reglas descritas en el apartado 7. Según éstas, aparecerán tres tablas una por cada entidad y otra por la relación. Por tanto el esquema relacional final será el siguiente:

ALUMNOS (dni, nombre, apellido, domicilio)
ASIGNATURAS (nombreasig, curso)
ALUM-ASIG (dni, nombreasig, nota)

♦ Problema 1.2

Se desea diseñar una base de datos para una Universidad que contenga información sobre los alumnos, las asignaturas y las carreras que se pueden

estudiar. Construir un modelo E-R y pasarlo posteriormente a un esquema relacional teniendo en cuenta las siguientes restricciones:

- Un alumno puede estar matriculado en muchas asignaturas
- Una asignatura sólo puede pertenecer a una sola carrera.
- Una carrera puede tener muchas asignaturas.

Solución:

Tal y como aparece en el enunciado del problema, existen tres tipos de entidades sobre las cuales se desea almacenar información: ALUMNOS, ASIGNATURAS y CARRERAS. Además, es fundamental saber las calificaciones obtenidas por cada alumno en cada asignatura, por lo que es necesario establecer una relación (AL-AS) entre estos dos conjunto de entidades. También se desea conocer cuáles son las asignaturas que pertenecen a cada carrera, por lo que es también necesario establecer otra relación (AS-CAR) entre los conjuntos de entidades ASIGNATURAS y CARRERAS.

Como puede verse, siempre que se necesita establecer una correspondencia entre dos conjuntos de entidades, es necesario incluir una relación con o sin atributos propios.

Siguiendo con el desarrollo del problema, antes de comenzar a dibujar el modelo E-R es necesario conocer qué conjuntos de atributos funcionarán en cada entidad como clave primaria.

ALUMNOS (dni, nombre, apellido, domicilio)
ASIGNATURAS (regasig, nomasig)
CARRERAS (nomcarr, duración)

Es interesante detenerse brevemente a pensar sobre los atributos que se han introducido en cada entidad. En el caso de la entidad ALUMNOS, no existe ninguna complicación adicional a la hora de elegir los mismos, es decir, el *dni* del alumno, su *nombre*, *domicilio* etc. Además, queda también claro que la manera de distinguir a un alumno de otro, es mediante su *dni*, por lo que ha sido este atributo el elegido como clave primaria. Podría también pensarse que la unión de los atributos *nombre* y *apellido*, funciona como clave aunque para ello no deberían existir dos alumnos con igual nombre y apellido (cosa un tanto improbable en una Universidad).

En el caso de la entidad ASIGNATURAS, ha sido necesario introducir un atributo adicional, denominado *regasig*, debido a que en una Universidad existen asignaturas distintas que se enmarcan en carreras distintas, pero que tienen el mismo nombre (*nomasig*). Por ejemplo "Matemáticas I" de CC. Económicas y "Matemáticas I" de CC. Biológicas. De esta manera, es posible diferenciar una asignatura de otra y cumplir con una de las condiciones de diseño que imponía que una asignatura sólo puede pertenecer a una carrera.

Para la entidad CARRERAS, se han introducido los dos atributos expuestos anteriormente y se ha supuesto, lo cual no es tan descabellado, que no existen dos carreras con el mismo nombre dentro de una misma Universidad.

Teniendo en cuenta todo esto, el diagrama E-R pedido es el siguiente:

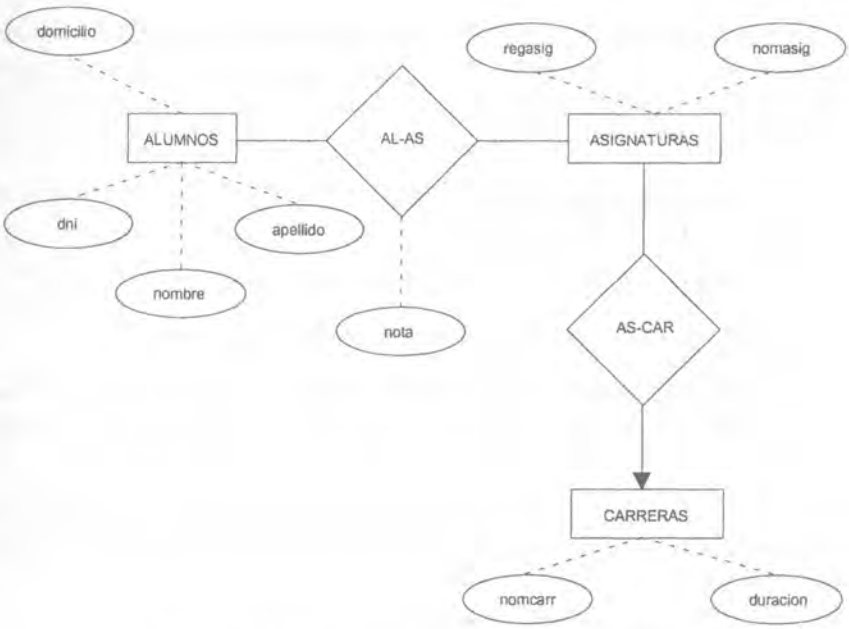


Figura 1.3: Diagrama E-R del problema 1.2

Según el modelo anterior y las reglas dadas en el apartado 7, aparecerán tres tablas relacionales debidas a las entidades en si, las cuales estarán compuestas por los atributos de las mismas, y dos tablas debidas a las relaciones, las cuales estarán compuestas por las claves primarias de las entidades que relacionan más los atributos propios de las relaciones.

Por tanto, las tablas finales serán:

ALUMNOS (dni, nombre, apellido, domicilio)
ASIGNATURAS (regasig, nomasig)
CARRERAS (nomcarr, duracion)
AL-AS (dni, regasig, nota)
AS-CAR (regasig, nomcarr)

♦ Problema 1.3

Se desea diseñar una base de datos para una Universidad que contenga información sobre los alumnos, las asignaturas y los profesores. Construir un modelo E-R y pasarlo posteriormente a un esquema relacional teniendo en cuenta las siguientes restricciones:

- Una asignatura puede estar impartida por muchos profesores (no a la vez) ya que pueden existir diversos grupos.
- Un profesor puede dar clases de muchas asignaturas.
- Un alumno puede estar matriculado en muchas asignaturas.
- Se necesita tener constancia de las asignaturas en las que está matriculado un alumno así como de las notas obtenidas y los profesores que le han calificado.
- También es necesario tener constancia de las asignaturas que imparten todos los profesores (independientemente de si tienen algún alumno matriculado en su grupo).
- No existen asignaturas con el mismo nombre.
- Un Alumno no puede estar matriculado en la misma asignatura con dos profesores distintos.

Solución:

Como en problemas anteriores, antes de plantear el modelo E-R se procederá a identificar las entidades con las cuales se va a trabajar y los atributos que funcionarán como claves primarias de las mismas.

Visto el enunciado del problema, se observa que existen tres conjuntos de entidades claramente definidos, los ALUMNOS, las ASIGNATURAS y los PROFESORES, a los cuales se les va a asociar los siguientes atributos (aunque se les podría añadir más, se va a trabajar sólo con los fundamentales):

ALUMNOS (dnia, noma, apella, domicila)
ASIGNATURAS (nomasig, curso)
PROFESORES (dnip, nomp, apellp, domicilp)

Ya que este es un problema académico, se han indicado sólo los atributos mínimos para que el diseño sea aceptable, es decir, para los alumnos su nombre (*noma*), su apellido (*apella*), su domicilio (*domicilia*) y el *dni*. Esto mismo se ha hecho con las otras dos entidades.

Conocidas las entidades que entrarán en juego en el diseño de la base de datos, es necesario ver ahora cuales serán las relaciones necesarias entre ellas para cumplir con las especificaciones del diseño.

Una de las condiciones, indica que se debe tener constancia de las asignaturas que imparten todos los profesores, por lo que es necesario establecer una relación que se denominará AS-PRO que asocie a los profesores con las asignaturas.

Por otra parte, otra de las especificaciones indica que también es necesario saber las asignaturas en las que está matriculado un alumno, así como la nota obtenida y el profesor que la imparte. Para ello se establecerá una relación ternaria, es decir, una relación que asocie a las tres entidades ASIGNATURAS, PROFESORES y ALUMNOS, con un atributo propio denominado *nota* y cuyo nombre será A-P-A.

Con todo esto, y teniendo en cuenta las cardinalidades de asignación impuestas por las condiciones de diseño, el diagrama E-R resultante es el que aparece en la Figura 1.4.

Es interesante destacar del diagrama, la flecha hacia la entidad profesor que indica que una pareja compuesta por un alumno y una asignatura, sólo pueden

estar asociadas a un único profesor, tal y como aparecía en las condiciones de diseño.

Falta ahora plasmar este diagrama en un esquema relacional, para lo cual se seguirán las mismas reglas aplicadas en los problemas anteriores es decir, aparecerán cinco tablas, tres debidas a las entidades y dos a las relaciones.

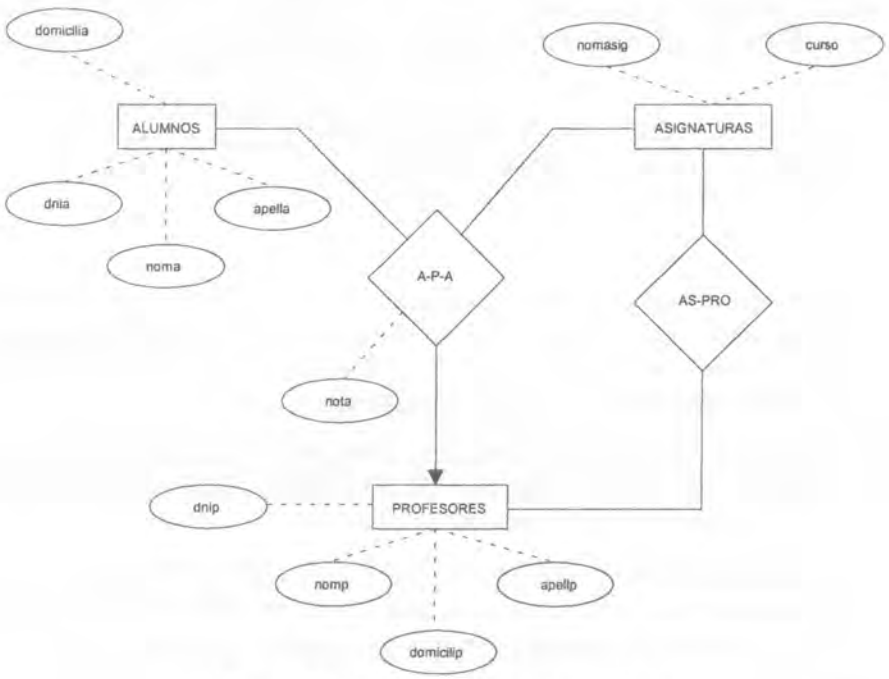


Figura 1.4: Modelo E-R del problema 1.3.

Como todas las entidades de las que se dispone son entidades fuertes, el paso al modelo relacional es extremadamente sencillo es decir, existirá una tabla por cada entidad con los atributos de la misma y otra tabla por cada relación la cual estará compuesta por las claves primarias de las entidades que relaciona, más los atributos propios de dichas relaciones. Aplicando lo anterior, las tablas resultantes serán las siguientes:

ALUMNO (dnia, noma, apella, domicilia)
ASIGNATURA (nomasig, curso)
PROFESOR (dnip, nomp, apellp, domicilp)
A-P-A (dnia, nomasig, dnip, nota)

AS-PRO (nomasig, dnip)

El resultado final, ha sido un diseño con 5 tablas relacionales las cuales satisfacen las condiciones impuestas a priori.

♦ Problema 1.4

Se desea diseñar una base de datos para una sucursal bancaria que contenga información sobre los clientes, las cuentas, las sucursales y las operaciones realizadas en cada una de las cuentas. Construir un modelo E-R y pasarlo posteriormente a un esquema relacional teniendo en cuenta las siguientes restricciones:

- Una operación viene determinada por su número de operación, la fecha y la cantidad.
- Un cliente puede tener muchas cuentas.
- Una cuenta puede pertenecer a varios clientes.
- Una cuenta solamente puede estar en una sucursal.

Solución:

Este es un problema típico, cuya solución introducirá un nuevo tipo de entidad que no se ha tratado hasta ahora. Además se verá cómo el mismo problema admite varios enfoques distintos y por tanto varias soluciones.

Teniendo en cuenta las consideraciones expuestas anteriormente, es fácil deducir que existirán los siguientes conjuntos de relaciones.

CLIENTES (dni, nombre, apellido)
SUCURSALES (numsuc, dirección)
CUENTAS (numcuent, tipo, interés, saldo)
OPERACIONES (numopera, fecha, cantidad)

Por supuesto, y como en casos anteriores, a cada una de ellas se le podría dotar de muchos más atributos, lo cual no supondría ninguna diferencia en cuanto a dificultad ni complejidad se refiere.

Antes de seguir, es importante llamar la atención sobre el hecho de que al conjunto de entidades OPERACIONES no se le ha asignado ninguna clave. Ello es debido a que aunque cada entidad del conjunto de entidades OPERACIONES es conceptualmente distinta, las operaciones en cuentas diferentes pueden compartir el mismo número de operación, por lo que no existe ninguna combinación de atributos que permita identificar de manera única a cada una de las entidades. OPERACIONES es por tanto una entidad débil.

Veamos la primera de las soluciones que presenta este problema.

Tal y como se cita en las condiciones de diseño, los clientes tienen cuentas y éstas deben pertenecer a una sola sucursal. Además, para cada cuenta debe quedar constancia de las operaciones realizadas, por lo que el primero de los modelos que da cuenta de ello sería el siguiente:

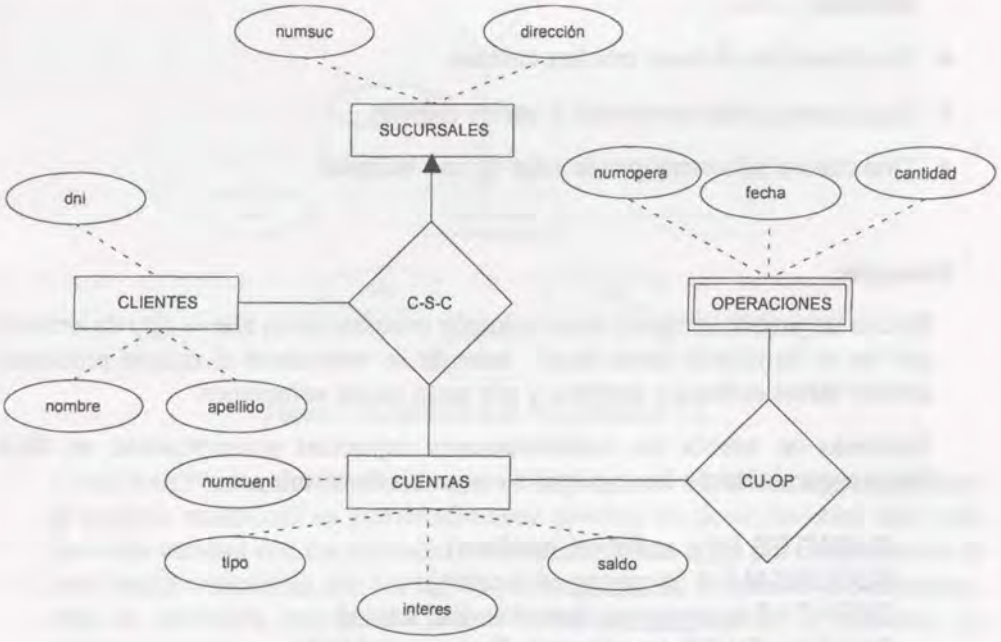


Figura 1.5: Primera solución del problema 1.4

Como puede verse, el modelo refleja el hecho de que cada cuenta, que por supuesto pertenece a uno o varios clientes, sólo puede pertenecer a una sucursal, es decir, cada pareja cliente y cuenta lleva asociada una y sólo una sucursal.

El diagrama también contempla que OPERACIONES es un conjunto de entidades débiles y que por lo tanto, sólo tiene sentido su existencia cuando van relacionadas a un conjunto de entidades fuertes que en este caso es CUENTAS.

Aplicando ahora las reglas del apartado 7, aparecerán cuatro tablas correspondientes a las cuatro entidades y otras dos más correspondientes a las dos relaciones existentes en el modelo.

Las tablas correspondientes a las entidades serán:

CLIENTES (dni, nombre, apellido)
SUCURSALES (numsuc, dirección)
CUENTAS (numcuent, tipo, interes, saldo)

Falta la tabla correspondiente al conjunto de entidades débiles OPERACIONES.

Calculando ahora las tablas correspondientes a las relaciones, es fácil ver que la tabla correspondiente a la relación C-S-C se calcula sin ningún problema tal y como se ha hecho hasta ahora, es decir:

C-S-C (dni, numsuc, numcuent)

Para calcular la tabla correspondiente a la relación CU-OP y al conjunto de entidades OPERACIONES, hay que tener en cuenta que estas últimas son débiles y que por tanto carecen de clave. Al carecer de ella y debido a que dependen de una entidad fuerte, es necesario asignarles una para así poder calcular cual va a ser la tabla que representa a la relación que une a ambas y la suya propia. Tal y como aparece en el apartado 4 de este capítulo, la clave primaria que se le asignará a una entidad débil estará compuesta por su *discriminador* y por la clave primaria del conjunto de entidades fuertes a los que está supeditada. El discriminador se definió anteriormente como el conjunto de atributos X de la entidad débil tal que, para cada valor de la clave primaria Y de la entidad fuerte a la que está supeditada, el valor de X identifica de manera única una entidad del conjunto de entidades débiles.

En este caso, el discriminador del conjunto de entidades OPERACIONES es el atributo *numopera*, ya que para cada cuenta un número de operación (*numopera*) identifica de forma única una única operación.

Por tanto las dos tablas que faltan correspondientes a la entidad débil y a la relación CU-OP serán, según las reglas del apartado 7:

OPERACIONES (numopera, fecha, cantidad, numcuent)
CU-OP (numcuent, numopera)

Resumiendo, el modelo Entidad-Relación dibujado anteriormente ha dado origen a las siguientes tablas relacionales:

CLIENTES (dni, nombre, apellido)
SUCURSALES (numsuc, dirección)
CUENTAS (numcuent, tipo, interes, saldo)
C-S-C (dni, numsuc, numcuent)
OPERACIONES (numopera, fecha, cantidad, numcuent)
CU-OP (numcuent, numopera)

Este no es el único modelo Entidad-Relación que tiene en cuenta todas las especificaciones de diseño expuestas en el problema. Otra posible solución sería la siguiente:

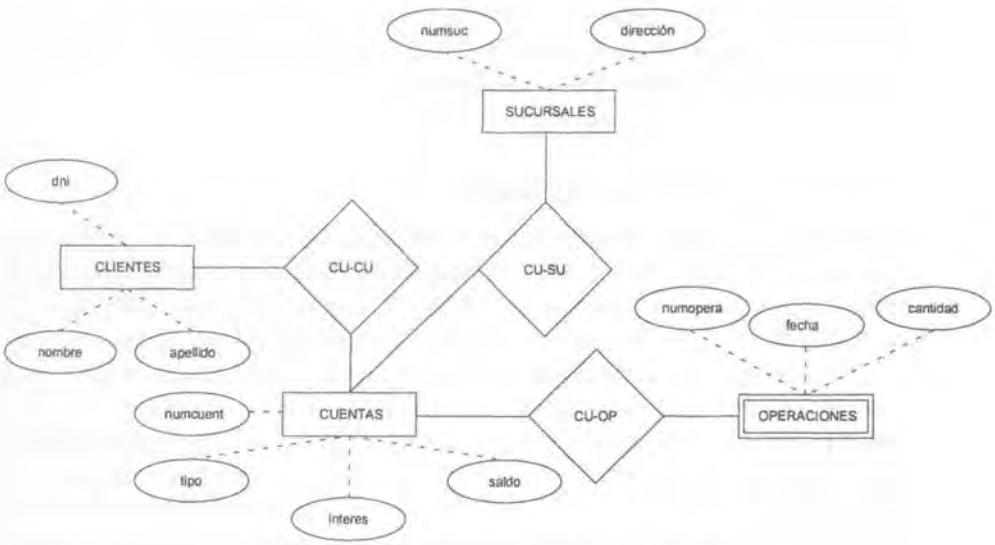


Fig 1.6: Segundo enfoque del problema 1.4

Este modelo cumple también las condiciones de diseño de la base de datos. La diferencia fundamental, es que ahora la relación ternaria C-S-C se ha deshecho en otras dos denominadas CLI-CU y CU-SU. Por decirlo de alguna manera, lo que se ha conseguido ha sido independizar el hecho de que un cliente posea una cuenta, con el de que ésta pertenezca a una determinada sucursal aunque

la información sigue siendo la misma es decir, en todo momento se pueden saber los clientes que pertenecen a una determinada sucursal o las sucursales en las que un cliente tiene cuentas abiertas.

Siguiendo los mismos pasos del desarrollo anterior, este modelo produciría ahora las siguientes tablas relacionales:

CLIENTES (dni, nombre, apellido)
SUCURSALES (numsuc, dirección)
CUENTAS (numcuent, tipo, interés, saldo)
CLI-CU (dni, numsuc)
CU-SU (numsuc, numcuent)
OPERACIONES (numopera, fecha, cantidad, numcuent)
CU-OP (numcuent, numopera)

Como puede verse, ha aparecido una tabla más que en el caso anterior debido a la independencia que existe ahora entre los clientes y las cuentas.

♦ Problema 1.5

Se desea diseñar una base de datos para un centro comercial organizado por departamentos que contenga información sobre los clientes que han comprado algo, los trabajadores, el género que se oferta y las ventas realizadas. Construir un modelo E-R y pasarlo posteriormente a un esquema relacional, teniendo en cuenta las siguientes restricciones.

- Existen tres tipos de trabajadores: gerentes, jefes y vendedores.
- Cada departamento está gestionado por un gerente.
- Un determinado producto sólo se encuentra en un departamento.
- Los jefes y vendedores sólo pueden pertenecer a un único departamento.
- Un gerente tiene a su cargo a un cierto número de jefes y éstos a su vez a un cierto número de vendedores.
- Una venta la realiza un vendedor a un cliente y debe quedar constancia del artículo vendido. Sólo un artículo por apunte de venta.

Solución:

Vistas las restricciones impuestas y las necesidades de la base de datos que se piensa diseñar, se pueden plantear a priori los siguientes conjuntos de entidades:

ARTICULOS (numgen, nombre, color, precio)
CLIENTES (dnicli, nombre, apellido, dirección)
GERENTES (dniger, nombre, apellido, dirección)
JEFES (dnijef, nombre, apellido, dirección)
VENDEDORES (dniven, nombre, apellido, dirección)
DEPARTAMENTOS (nomdep, piso)

Con estas entidades y con las especificaciones anteriores, se puede plantear el siguiente modelo E-R.

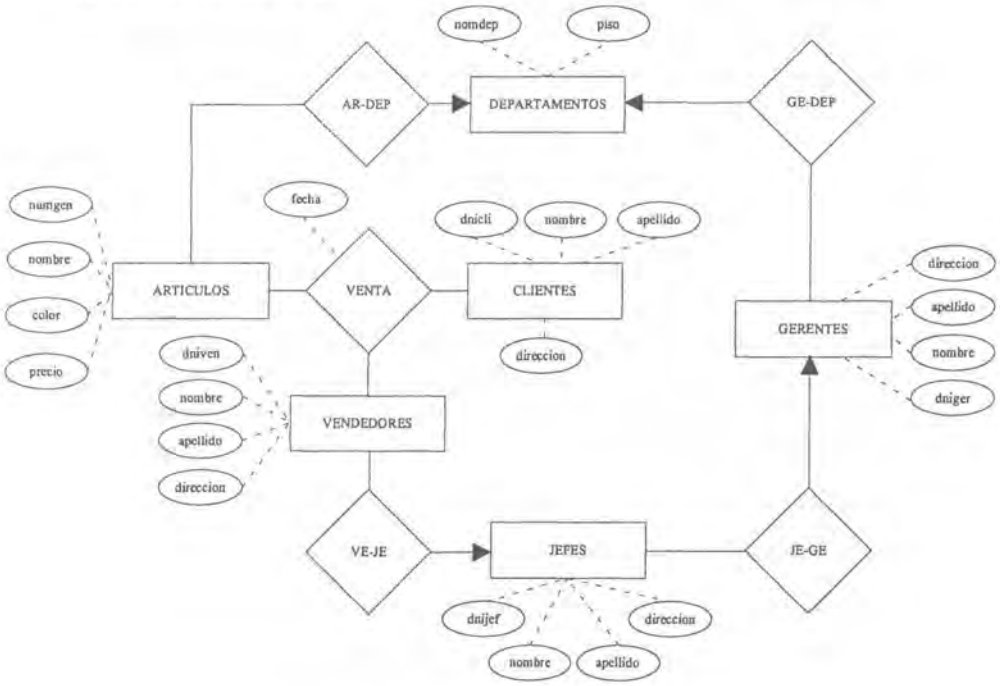


Figura 1.7: Solución al problema 1.5

Como puede verse, el modelo anterior cumple con todas las necesidades impuestas en el diseño tal y como reflejan las ventas realizadas, las cuales se han implementado como una relación entre los conjuntos de entidades CLIENTES, ARTICULOS y VENDEDORES con un atributo propio que expresa la fecha en la cual se realizó la misma.

También mediante la relación AR-DEP se ha expresado el hecho de que un cierto artículo pertenezca a un determinado departamento. Como puede observarse, la relación se ha modelado de muchos a uno, ya que un departamento tendrá muchos artículos pero un artículo solamente pertenecerá a un departamento.

Lo mismo sucede con las relaciones GE-DEP, JE-GE y VE-JE las cuales también están modeladas como muchas a una.

El siguiente paso, consiste en plasmar este esquema Entidad-Relación en un esquema relacional, aplicando para ello las reglas dadas en el apartado 7, las cuales nos indican que existirán 11 tablas relacionales, 6 correspondientes a las entidades y las otras 5 correspondientes a las relaciones entre las mismas, es decir:

ARTICULOS (numgen, nombre, color, precio)
CLIENTES (dnicli, nombre, apellido, dirección)
GERENTES (dniger, nombre, apellido, dirección)
JEFES (dnijef, nombre, apellido, dirección)
VENDEDORES (dniven, nombre, apellido, dirección)
DEPARTAMENTOS (nomdep, piso)
VENTA (numgen, dnicli, dniven, fecha)
AR-DEP (numgen, nomdep)
GE-DEP (dniger, nomdep)
JE-GE (dnijef, dniger)
VE-JE (dniven, dnijef)

Es interesante observar que la clave de la tabla VENTAS estará compuesta por sus cuatro atributos, debido a que sólo con los tres primeros es imposible identificar de manera única cada una de las tuplas, debido a que un cliente puede comprar el mismo artículo por segunda vez al mismo vendedor pero en fechas distintas.

Este diseño anterior no es el único que refleja todas las características de la base de datos, ya que existen otros esquemas cuyo nivel de información es el mismo.

Antes de proseguir, se introducirá un concepto nuevo denominado *generalización*. En muchas ocasiones, cuando se realiza un modelo Entidad-Relación, existen conjuntos de entidades que comparten un determinado número de atributos como por ejemplo:

POLICIAS (dni, nombre, apellido, graduación, comisaria)
BARRENDEROS (dni, nombre, apellido, distrito, ciudad)

Es fácil ver que estos dos conjuntos de entidades tienen en común los tres primeros atributos (dni, nombre, apellido) y se diferencian en los dos últimos. Si estos dos conjuntos de entidades apareciesen en un mismo modelo E-R, por ejemplo en una base de datos de un ayuntamiento, sería una buena idea establecer un tipo de conjuntos de entidades superior que se podría denominar EMPLEADOS que tuviese como atributos (dni, nombre, apellido) para que de esta manera los conjuntos de entidades POLICIAS y BARRENDEROS fueran clases especializadas de los conjuntos de entidades EMPLEADO.

Pues bien, en términos de un diagrama Entidad-Relación, esta técnica se denomina generalización y se representa mediante un triángulo etiquetado ISA ("is a") tal y como puede observarse en la siguiente figura:

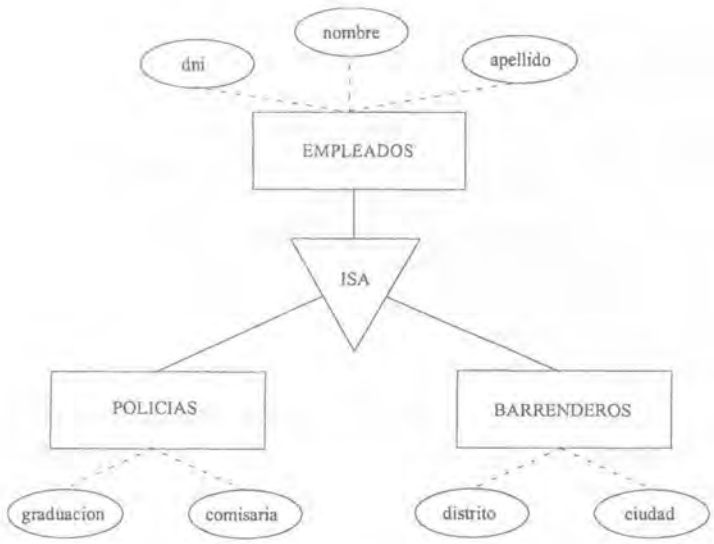


Figura 1.8: Generalización

Este esquema está expresando que existen dos tipos de conjuntos de entidades denominados POLICIAS y BARRENDEROS, cada una de ellas con cinco atributos, dos propios y otros tres comunes y heredados del conjunto de entidades superior EMPLEADOS.

De esta manera se aplica parte de la filosofía del diseño orientado a objeto al desarrollo de modelos E-R.

Existen dos métodos diferentes para transformar un diagrama E-R que incluye generalización a una forma tabular:

- Crear una tabla para el conjunto de entidades de nivel más alto, tal y como aparece en la sección 1.7. Para cada conjunto de entidades de nivel más bajo, crear una tabla que incluya una columna para cada uno de los atributos de ese conjunto de entidades, más una columna para cada atributo de la clave primaria del conjunto de entidades del nivel más alto.

  Para el caso anterior, se crearían las siguientes tablas relacionales:

  EMPLEADO (dni, nombre, apellido)
  POLICIAS (dni, graduación, comisaría)
  BARRENDEROS (dni, distrito, ciudad)

  El otro método es el siguiente:

- Para el conjunto de entidades de nivel más alto, no crear ninguna tabla; y para cada conjunto de entidades de nivel más bajo, crear una tabla, que incluya una columna para cada uno de los atributos de ese conjunto de entidades más una columna para cada uno de los atributos del conjunto de entidades del nivel más alto.

Utilizando este método, el ejemplo anterior quedaría:

POLICIAS (dni, nombre, apellido, graduación, comisaría)
BARRENDEROS (dni, nombre, apellido, distrito, ciudad)

Como puede verse, el conjunto de datos obtenido sigue siendo el mismo que con el método anterior, aunque organizados de otra manera. En cuanto a cual de los dos métodos es mejor, la respuesta es que cualquiera de ellos proporciona resultados satisfactorios.

Terminada esta introducción a la generalización, procederemos ahora a resolver el problema propuesto haciendo uso de dicha técnica (si es que se presta a ello).

Vistas las características de la base de datos que se va a diseñar y dadas como en el caso anterior los siguientes conjuntos de entidades:

ARTICULOS (numgen, nombre, color, precio)
CLIENTES (dnicli, nombre, apellido, dirección)
GERENTES (dniger, nombre, apellido, dirección)
JEFES (dnijef, nombre, apellido, dirección)
VENDEDORES (dniven, nombre, apellido, dirección)
DEPARTAMENTOS (nomdep, piso)

Es fácil ver que los conjuntos de entidades GERENTES, JEFES y VENDEDORES tienen todos los atributos iguales, por lo que de alguna manera se puede pensar en un conjunto se entidades de nivel más alto que podríamos denominar EMPLEADOS cuyos atributos serían:

EMPLEADOS (dni, nombre, apellido, dirección)

de la cual las anteriores serían subentidades.

Por tanto, teniendo en cuenta lo anteriormente expuesto, podría plantearse el siguiente modelo E-R:

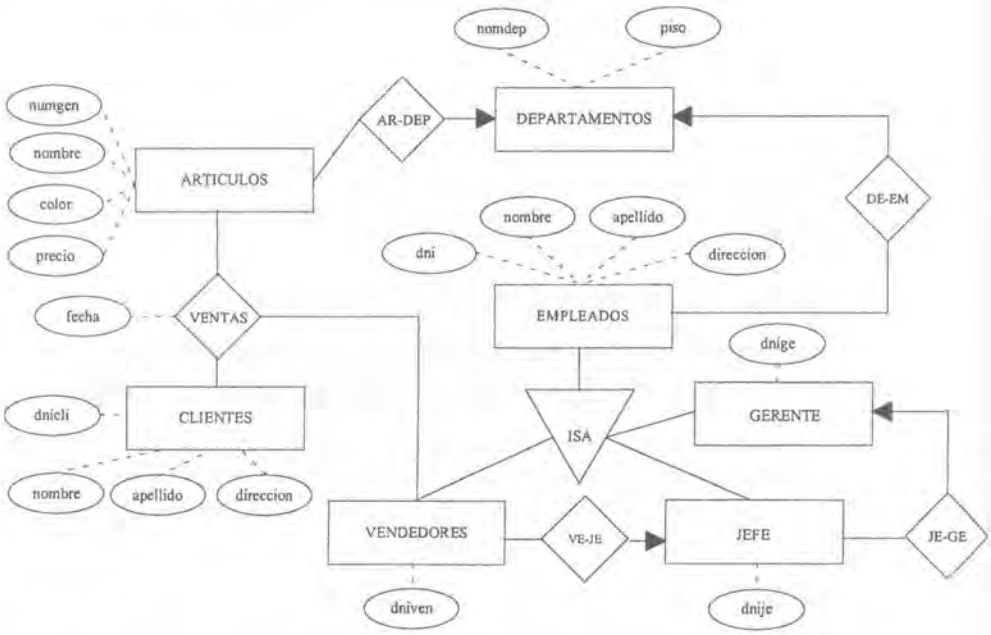


Figura 1.9: Solución al problema 1.5 utilizando generalización

Es interesante detenerse un momento para comentar el diseño anterior:

- Como puede observarse de la figura 1.9, se ha establecido un conjunto de entidades superior denominada EMPLEADOS del cual cuelgan los otros dos conjuntos de entidades a las cuales se les ha añadido un atributo propio (dniven, dnije y dnige) para poder establecer relaciones entre ellas.
- Aunque el conjunto de entidades CLIENTES tiene a priori los mismos atributos que el conjunto de entidades EMPLEADOS, no se ha incluido dentro de este grupo porque conceptualmente pertenecen a clases distintas.

Posteriormente se planteará el modelo E-R incluyendo a CLIENTES dentro de la clase EMPLEADOS.

El siguiente paso, consiste en plasmar este modelo Entidad-Relación en un esquema relacional, aplicando para ello las reglas dadas en el apartado 7, las cuales indican que existirán 12 tablas relacionales (una más que antes), 7 correspondientes a las entidades y las otras 5 correspondientes a las relaciones entre las mismas, es decir:

ARTICULOS (numgen, nombre, color, precio)
CLIENTES (dnicli, nombre, apellido, direccion)
EMPLEADOS (dni, nombre, apellido, direccion)
GERENTES (dniger)
JEFES (dnijef)
VENDEDORES (dniven)
DEPARTAMENTOS (nomdep, piso)
VENTAS (numgen, dnicli, dniven, fecha)
AR-DEP (numgen, nomdep)
GE-DEP (dniger, nomdep)
JE-GE (dnijef, dniger)
VE-JE (dniven, dnijef)

El lector habrá notado que existen ciertas tablas en las cuales faltan atributos, ya que si se aplican de manera estricta las reglas sobre la generalización vistas anteriormente, las tablas VENDEDORES, JEFES y GERENTES deberían presentar el siguiente aspecto:

GERENTE (dniger, dni)
JEFE (dnijef, dni)
VENDEDOR (dniven, dni)

y no el que se ha elegido como resultado final, es decir:

GERENTES (dniger)
JEFES (dnijef)
VENDEDORES (dniven)

Lo que ocurre es que como el valor de los dos atributos que componen cada una de las tres tablas es el mismo, se ha optado por suprimir uno de los dos sin que ello suponga pérdida alguna de información.

Antes se ha comentado el hecho de que el conjunto de entidades CLIENTES también tiene atributos comunes con los conjuntos de entidades

VENDEDORES, JEFES y GERENTES, por lo que también se puede pensar en aplicar la generalización a este conjunto de entidades y hacer que dependan también de un conjunto de entidades superior que podría denominarse ahora PERSONAS con los siguientes atributos:

PERSONAS (dni, nombre, apellido, domicilio)

Teniendo en cuenta esto, podría entonces plantearse el siguiente modelo E-R:

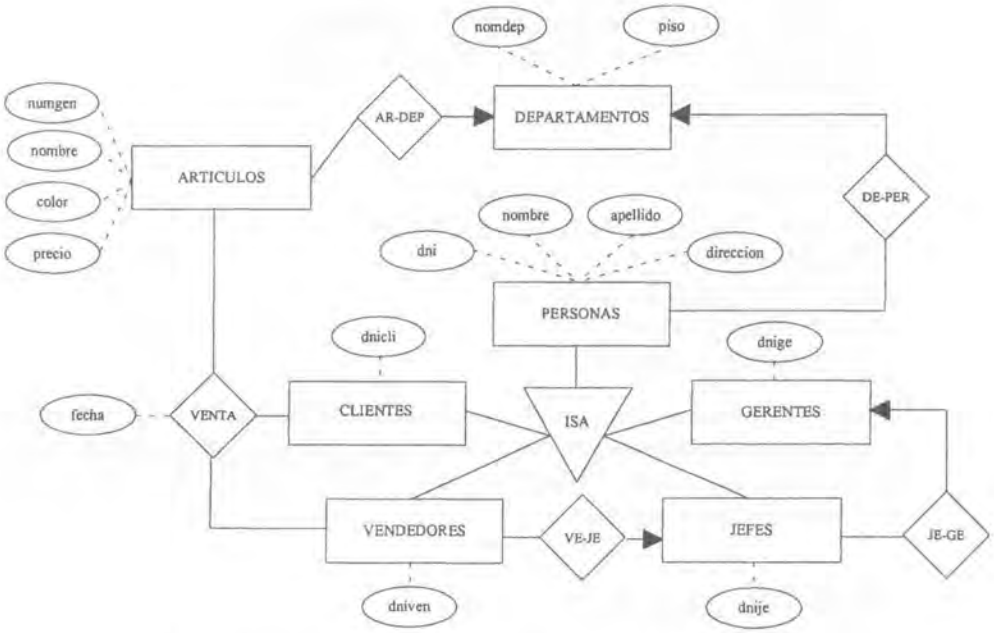


Figura 1.10: Segunda solución a 1.5 utilizando generalización

el cual, aplicando las mismas reglas que en el caso anterior, produce el siguiente conjunto de tablas relacionales:

ARTICULOS (numgen, nombre, color, precio)
CLIENTES (dnicli)
PERSONAS (dni, nombre, apellido, domicilio)
GERENTES (dniger)
JEFES (dnijef)

VENDEDORES (dniven)
DEPARTAMENTOS (NomDep, Piso)
VENTAS (numgen, dnicli, dniven, fecha)
AR-DEP (numgen, nomdep)
GE-DEP (dniger, nomdep)
JE-GE (dnijef, dniger)
VE-JE (dniven, dnijef)

que como puede observarse, el sistema obtenido cumple también perfectamente las especificaciones del problema.

Cabe ahora preguntarse: ¿Cuál de todos los modelos es el mejor?. Probablemente la pregunta podría contestarse diciendo que los tres, ya que como además se ha comprobado, el resultado final obtenido en cualquiera de los casos es muy parecido.

♦ Problema 1.6

Se desea diseñar una base de datos para una DISCOTECA-VIDEOTECA que contenga información de vídeos, discos, socios, empleados y préstamos. Construir un modelo E-R y pasarlo posteriormente a un esquema relacional teniendo en cuenta las siguientes restricciones.

- Un socio puede tener en préstamo varios vídeos y discos a la vez.
- Un video o disco sólo puede estar prestado a un socio.
- Un empleado puede prestar muchos discos y vídeos.
- Cuando se realiza un préstamo deben aparecer el socio, el vídeo o disco, la fecha y el empleado.
- En los discos debe aparecer información sobre su autor y en los vídeos su protagonista.

Añada los campos y la información que crea necesaria para que el diseño sea correcto.

Solución:

Tal y como aparece en las condiciones de diseño, la base de datos deberá presentar los siguientes conjuntos de entidades básicas:

EMPLEADOS (dni, nombre, apellidos, domicilio)
SOCIOS (numsoc, dni, nombre, apellidos, domicilio)
VIDEOS (numreg, titulo, protagon)
DISCOS (numreg, titulo, autor)

Los atributos asignados a las entidades anteriores son los habituales, siendo el *numreg*, el número de registro que lleva cada cinta de vídeo o disco y que es el mismo para todas las cintas o discos que tengan el mismo título. Es decir, si en el videoclub se tienen tres cintas de 'La Guerra de las Galaxias', todas llevan el mismo *numreg* (esto es simplemente una suposición).

En principio, parece que las dos primeras entidades no presentan ningún problema ya que reúnen toda la información necesaria acerca de los entes que representan. Sin embargo a las dos entidades siguientes, les falta algo para que puedan ser identificadas de manera única ya que, como se ha comentado antes, todas las películas con el mismo título presentan el mismo *numreg*.

Existen varias soluciones para abordar este problema. La primera consiste en añadir a cada una de las dos tablas, un campo nuevo denominado *numcop* que especifique de manera única a una cinta o disco con un cierto *numreg*.

Por ejemplo, siguiendo el caso anterior, cada una de las cintas de 'La Guerra de las Galaxias' tendría un *numcop* distinto, el 1, 2 y 3. De esta manera, una cinta cualquiera estará especificada por su *numreg* y su *numcop*. Los conjuntos de entidades que entonces resultarían junto con sus claves primarias serían ahora:

EMPLEADOS (dni, nombre, apellidos, domicilio)
SOCIOS (numsoc, dni, nombre, apellidos, domicilio)
VIDEOS (numreg, numcop, titulo, protagon)
DISCOS (numreg, numcop, titulo, autor)

El modelo E-R que podría plantearse con las especificaciones anteriores sería el que se muestra en la figura 1.11.

Como puede verse, se ha utilizado también en este caso la generalización dado que el conjunto de entidades de vídeos y de discos tienen dos atributos en común. También podría hacerse lo mismo entre los empleados y los socios, pero se ha creído más conveniente realizarlo de esta manera.

Aplicando ahora las reglas que se han venido dando a lo largo del capítulo, el conjunto de tablas resultante será el siguiente:

EMPLEADOS (dni, nombre, apellidos, domicilio)
SOCIOS (numsoc, dni, nombre, apellidos, domicilio)
VIDEO-DISCO (numreg, numcop)
VIDEOS (numreg, numcop, titulo, protagon)
DISCOS (numreg, numcop, título, autor)
ALQUILER (numreg, numcop, numsoc, dni, fecha)

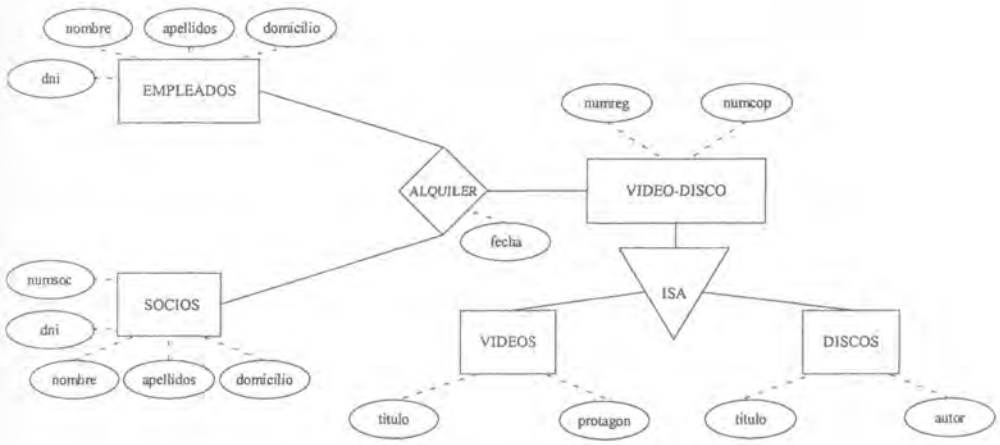


Figura 1.11: Solución al problema 1.6

Llegados a este punto, el lector seguramente habré advertido que aunque el diseño es formalmente correcto, algunas de las tablas presentan una redundancia de la información nada deseable. Este es el caso de VIDEOS y DISCOS.

Es fácil darse cuenta que por cada copia que exista de un cierto título, se está repitiendo en estas tablas toda la información restante (*titulo, ...*) la cual es única para cada *numreg*. Este problema, como se estudiará en capítulos posteriores, se debe a que dichas tablas no cumplen la segunda forma normal 2FN, regla que en cierta medida garantiza la no existencia de redundancia en las tablas. Como solución a ello, existen mecanismos perfectamente formalizados que permiten tratar este problema y conseguir unas tablas totalmente normalizadas. Estas reglas serán estudiadas en el capítulo 8.

Por ello, se va a proponer otro modelo E-R del problema que por el hecho de no usar la generalización, obtiene tablas libres de este fallo.

Partiendo de los mismos conjuntos de entidades básicos se propone ahora el siguiente modelo:

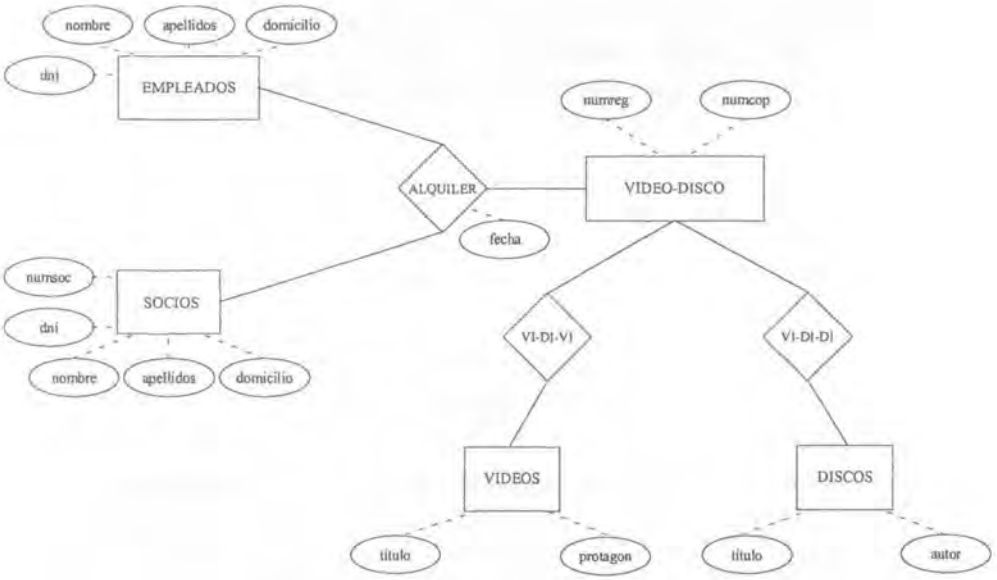


Figura 1.12: Segunda solución del problema 1.6

Como puede observarse en la figura anterior, se ha partido ahora de un sólo conjunto de entidades básico denominado VIDEO-DISCO el cual engloba todo el material existente en el videoclub. Para determinar ahora si un determinado *numreg* de una entidad perteneciente al conjunto de entidades VIDEO-DISCO, pertenece al conjunto de entidades de VIDEOS o de DISCOS, se han planteado dos relaciones una con cada uno de estos últimos conjuntos de manera que además a cada título de vídeo o de disco, se le asigne un *numreg*.

Aplicando ahora las reglas definidas en teoría, las tablas resultantes serán:

EMPLEADOS (dni, nombre, apellidos, domicilio)
SOCIOS (numsoc, dni, nombre, apellidos, domicilio)
VIDEO-DISCO (numreg, numcop)
VIDEOS (titulo, protagon)

DISCOS (título, autor)
VI-DI-VI (numreg, titulo, protagon)
VI-DI-DI (numreg, titulo, autor)
ALQUILER (numreg, numcop, numsoc, dni, fecha)

♦ Problema 1.7

Se desea diseñar una base de datos para una agencia matrimonial que contenga información de hombres (con todos sus datos personales), mujeres (con todos sus datos personales), empleados (divididos en tres categorías socios, directores y administrativos), citas realizadas (debe quedar constancia de la fecha, el hombre, la mujer y el director que la promovió) y matrimonios (fecha, hombre, mujer). Construir un modelo E-R y pasarlo posteriormente a un esquema relacional teniendo en cuenta las siguientes restricciones.

- Un hombre puede tener citas con varias mujeres.
- Una mujer puede tener citas con varios hombres.
- Un hombre puede casarse con varias mujeres ya que puede enviudar o divorciarse.
- Una mujer puede casarse con varios hombres ya que puede enviudar o divorciarse.
- Sólo los directores pueden promover citas.
- Un socio tiene a su cargo varios directores y éstos a su vez varios administrativos.

Solución:

Tal y como aparece en las condiciones de diseño, la base de datos deberá presentar las siguientes entidades básicas:

HOMBRES (dni, nombre, apellidos, domicilio)
MUJERES (dni, nombre, apellidos, domicilio)
SOCIOS (dni, nombre, apellidos, domicilio)
DIRECTORES (dni, nombre, apellidos, domicilio)
ADMINISTRATIVOS (dni, nombre, apellidos, domicilio)

las cuales, como puede verse, comparten el mismo conjunto de atributos.

No obstante, es fácil darse cuenta de que entre las cinco entidades presentes, existen dos grupos claramente definidos: los que trabajan en la agencia y los clientes de la misma. Sería entonces una buena idea utilizar el concepto de generalización para los empleados de la empresa y mantener separadas las otras dos entidades, ya que entre ellas se deberán establecer algunas relaciones no solo entre si sino además con el conjunto de entidades DIRECTORES.

Por ello, se puede proponer el diseño mostrado en la Figura 1.13 para modelar la base de datos en cuestión.

Como se ha hecho en problemas anteriores, al atributo *dni* de los conjuntos de entidades HOMBRES y MUJERES se le ha añadido un carácter más (*dnih* y *dnim*) para que al establecer las relaciónes BODAS y CITAS no aparezcan dos atributos con el mismo nombre.

Aplicando las reglas vistas a lo largo de este capítulo, es fácil ver que el conjunto de tablas relacionales resultantes es el siguiente:

HOMBRES (dnih, nombre, apellidos, domicilio)
MUJER (dnim, nombre, apellidos, domicilio)
EMPLEADOS (dni, nombre, apellidos, domicilio)
ADMINISTRA (dnia)
DIRECTORES (dnid)
SOCIOS (dnis)
AD-DIR (dnia, dnid)
DIR-SOC (dnid, dnis)
CITAS (dnih, dnim, dnid, fecha)
BODAS (dnih, dnim)

El lector habrá notado que, al igual que en el problema 5, existen ciertas tablas en las cuales faltan atributos, ya que si se aplican de manera estricta las reglas sobre la generalización de la página 20, las tablas ADMINISTRA, DIRECTORES y SOCIOS deberían presentar el siguiente aspecto:

ADMINISTRA (dnia, dni)
DIRECTORES (dnid, dni)
SOCIOS (dnis, dni)

y no el que se ha elegido como resultado final, es decir:

ADMINISTRA (dnia)
DIRECTORES (dnid)
SOCIOS (dnis)

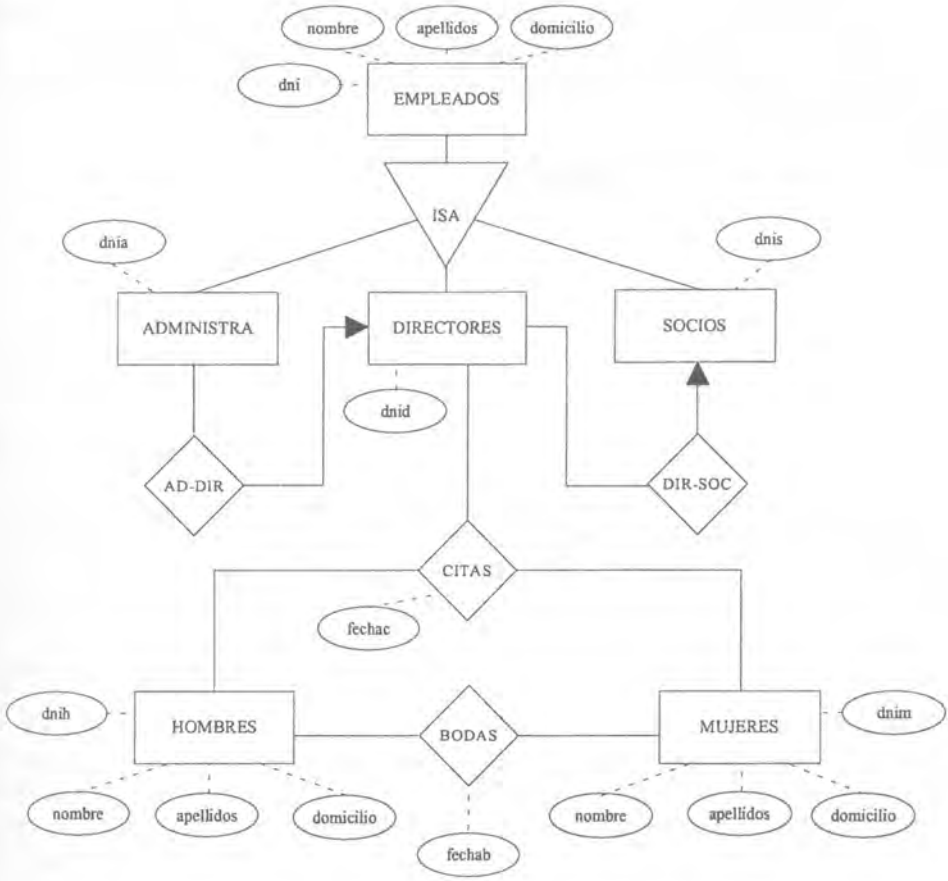


Figura 1.13: Solución al problema 1.7

Lo que ocurre es que como el valor de los dos atributos que componen cada una de las tres tablas es el mismo, se ha optado por suprimir uno de los dos sin que ello suponga pérdida alguna de la información.

Para terminar este capítulo, y como comentario final, es importante sacar en claro que el modelo E-R es una mera herramienta de diseño de bases de datos, que en ningún momento asegura que éste sea el más correcto posible. Sólo el buen criterio y la práctica a la hora de utilizar este método hará que los resultados obtenidos sean del todo satisfactorios. Otro punto que todavía quedaría por discutir sería el de si en el caso de que el modelo E-R se utilice para diseñar bases de datos relacionales, las tablas obtenidas son correctas desde el punto de vista formal. La respuesta a esto es en principio NO, es

decir, aunque el modelo exprese de manera correcta la información que se desea incluir en la base de datos, las tablas obtenidas pueden necesitar ser normalizadas para asegurar que no se producirán errores de inconsistencia ni de incongruencia de la información. Este tema será tratado en el capítulo 8.

Capítulo 2
Álgebra Relacional

2.1 Introducción

De forma general, una *base de datos* es un conjunto de *datos* relacionados entre sí, entendiendo por *datos*, hechos conocidos que pueden registrarse y que tienen un significado implícito. Por ejemplo, los nombres y direcciones de personas que pertenecen a un determinado colectivo. Las bases de datos están dirigidas a grupos de usuarios, quienes las utilizan para un propósito específico, y en ellas se representan aspectos del mundo real y se reflejan sus cambios.

Un procedimiento para *representar* los datos y *manipular* dicha representación lo proporciona el *modelo relacional*, quien se ocupa también de la *integridad* (exactitud o validez de los datos). En el modelo relacional los datos se representan mediante *relaciones* y un procedimiento para su manipulación lo proporciona el *álgebra relacional*.

En este capítulo se proponen y resuelven problemas de álgebra relacional y se realiza un recordatorio de los aspectos teóricos más importantes en relación con los problemas. La teoría se ha dividido en dos partes dedicadas a la *estructura de datos relacional* y a la manipulación de los datos mediante el *álgebra relacional*.

2.2 Estructura de datos relacional

Como se apuntó en la introducción, uno de los aspectos relativos a los datos de los que se ocupa el modelo relacional es el de su representación o estructura.

Una *base de datos relacional* se representa mediante un conjunto de *tablas*, a cada una de las cuales se le asigna un nombre exclusivo. Cada tabla, como se muestra en la figura 2.1, está formada por *filas* y *columnas* y en ella se encuentran sus datos. En dicha figura también se observa que a cada columna se le asigna un nombre. Por otra parte, todos los datos que aparecen en una misma columna deben

pertenecer a un mismo *dominio*, entendiéndose por dominio, el conjunto de valores permitidos para los datos que aparecen en cada una de las columnas.

	nombre 1	*nombre 2*	...	*nombre n*
Fila 1 →	dato 1,1	dato 1,2	...	dato 1,n
Fila 2 →	dato 2,1	dato 2,2	...	dato 2,n
... →	...	...	...	...
Fila m →	dato m,1	dato m,2	...	dato m,n
	↑	↑	↑	↑
	Columna 1	Columna 2	...	Columna n

Figura 2.1: Tabla

Si D_1, D_2, ..., D_n denotan los dominios correspondientes a cada una de las n columnas de la tabla, respectivamente, sus filas serán *tuplas* $< d_1, d_2, ..., d_n >$, donde los valores d_i pertenecen a los dominios D_i. En tal caso, se puede expresar el *producto cartesiano* de los distintos dominios como:

$$D_1 \times D_2 \times ... \times D_n = \{< d_1, d_2, ..., d_n > \mid d_1 \in D_1 \wedge d_2 \in D_2 \wedge ... \wedge d_n \in D_n\}$$

Es decir, el producto cartesiano $D_1 \times D_2 \times ... \times D_n$ es un conjunto cuyos elemento son todas las posibles tuplas $< d_1, d_2, ..., d_n >$.

Por tanto, el contenido de una tabla será un subconjunto del producto cartesiano que se conoce con el nombre de *relación*.

Como las tablas son esencialmente relaciones, en el modelo relacional se utilizan los términos *relación* y *tupla* en lugar de los términos *tabla* y *fila*, respectivamente. Por otra parte, en lugar de *columna* se utiliza el término formal *atributo*.

	nombre 1	*nombre 2*	...	*nombre n*
Tupla 1 →	dato 1,1	dato 1,2	...	dato 1,n
Tupla 2 →	dato 2,1	dato 2,2	...	dato 2,n
... →	...	...	...	...
Tupla m →	dato m,1	dato m,2	...	dato m,n
	↑	↑	↑	↑
	Atributo 1	Atributo 2	...	Atributo n

Figura 2.2: Relación

Teniendo en cuenta esta terminología la figura 2.1 pasa a la forma indicada en la figura 2.2. A modo de resumen, en la tabla de la figura 2.3, se indica la terminología formal que se utiliza en la estructura de datos relacional y la terminología informal equivalente.

TÉRMINO FORMAL	TÉRMINO INFORMAL
Relación	tabla
Atributo	columna o campo
Grado	número de columnas
Tupla	fila o registro
Cardinalidad	número de filas
clave primaria	identificador único
Dominio	fondos de valores legales

Figura 2.3: Terminología

En términos formales, una *relación* se compone de dos partes:

- *Cabecera* o conjunto de atributos.
- *Cuerpo* o conjunto de tuplas.

El *grado n* (número de atributos) de una relación permanece fijo en el tiempo mientras que su *cardinalidad m* (número de tuplas) varía con éste.

Por otra parte, una relación cumple las siguientes propiedades:

- No tiene tuplas repetidas.
- Las tuplas no están ordenadas.
- Los atributos no están ordenados.
- Todos los valores de los atributos simples son *atómicos*.

Cada una de las relaciones que forma parte de un *sistema relacional* puede ser de alguno de los siguientes tipos:

- *Relación base*. Relación con nombre que forma parte de la base de datos.
- *Vista*. Relación virtual con nombre definida en función de otras relaciones con nombre. No posee datos propios almacenados.
- *Instantánea*. Relación con nombre definida en función de otras relaciones con nombre. Posee datos propios almacenados.
- *Resultado de consulta*. Relación con o sin nombre resultante de alguna consulta especificada.
- *Resultado intermedio*. Relación con o sin nombre resultante de alguna expresión relacional anidada dentro de otra expresión.

- *Relación temporal.* Relación con nombre que se destruye en algún momento sin que el usuario realice ninguna acción para ello.

Por otra parte, se define como *clave primaria* de una relación, a uno o varios de sus atributos en los que no existen dos tuplas con el mismo valor en dicho atributo o atributos. La clave primaria es un identificador único para la relación.

Una relación puede tener más de un identificador único o *claves candidatas*. Uno o varios atributos de una relación forman una clave candidata si, y sólo si, se satisfacen las dos propiedades siguientes que no dependen del tiempo:

- *Unicidad.* En cualquier momento no existen dos tuplas en la relación con el mismo valor del atributo o atributos.
- *Minimalidad.* Para el caso de varios atributos, no será posible eliminar ninguno de ellos sin destruir la propiedad de unicidad.

Respecto a la clave primaria se pueden hacer los siguientes comentarios:

- Del conjunto de las claves candidatas se debe elegir una, y sólo una, clave primaria.
- Una *clave alternativa* es una clave candidata que no es clave primaria.
- Toda relación tiene por lo menos una clave candidata.
- El razonamiento para elegir la clave primaria, cuando existen varias claves candidatas, queda fuera del modelo relacional.
- La clave primaria es la que tiene verdaderamente importancia.
- La clave primaria es un concepto aplicable a las relaciones base.
- La clave primaria se define por el conjunto de atributos que la componen.
- El único modo de localizar una tupla es por el valor de su clave primaria.

Como se ha indicado, un *dominio* es una colección de valores de los cuales uno o más atributos obtienen sus valores reales. En relación con los dominios se puede resaltar lo siguiente:

- Las comparaciones entre atributos de distintos dominios no tienen sentido.
- Los dominios pueden ser simples o compuestos.
- Un dominio corresponde a *tipo de datos* en un lenguaje de programación.

Para finalizar, se define una *base de datos relacional* como una base de datos percibida por el usuario como una colección de relaciones normalizadas de diversos grados que varía con el tiempo.

2.3 Álgebra relacional

El álgebra relacional es un *lenguaje de consulta procedimental*. Un *lenguaje de consulta* es un lenguaje mediante el cual el usuario solicita información contenida en la base de datos. Este tipo de lenguajes se pueden clasificar en *lenguajes procedimentales* y *lenguajes no procedimentales*.

En un lenguaje procedimental, el usuario indica la secuencia de operaciones que se debe realizar sobre la base de datos para obtener el resultado deseado. Por otra parte, en un lenguaje no procedimental, el usuario describe la información que desea obtener de la base de datos sin dar el procedimiento para determinarla.

Para realizar las consultas de la base de datos, el álgebra relacional dispone de un conjunto de *operaciones* y de la *operación de asignación*. Esta última asigna el valor de alguna expresión del álgebra a una relación nombrada. Por otra parte, cada operación toma una o dos relaciones como entrada y produce una nueva relación como salida.

En el álgebra relacional se definen ocho operaciones básicas clasificadas en dos grupos:

- *Operaciones de conjuntos*: unión, intersección, diferencia y producto cartesiano.
- *Operaciones relacionales*: selección, proyección, reunión (natural y theta) y división.

Además de estas ocho operaciones básicas se define la *operación renombrar*, que permite cambiar de nombre los atributos, y otras *operaciones adicionales* (ampliación, resumen y división generalizada). Las ocho operaciones básicas no constituyen un conjunto mínimo. Así, las operaciones reunión, intersección y división se pueden definir a partir de las otras cinco.

2.3.1 Operación renombrar

En una relación, los nombres de los atributos que forman su cabecera deben ser distintos. Esto no impide que, en una base de datos relacional, pueda existir más de una relación con un mismo nombre para alguno de sus atributos. En tal caso, si mediante una operación se llega a otra relación que los contenga, se estaría incumpliendo la exigencia indicada.

Así, si R_1 y R_2 son dos relaciones en las que un atributo de R_1 y otro atributo de R_2 tienen el mismo nombre, y se obtiene una nueva relación con los atributos de R_1 y los atributos de R_2, en ésta aparecerán dos atributos con nombres iguales. Para evitar esta duplicidad se introduce la operación *renombrar*, cuya misión es cambiar de nombre los atributos que sean necesarios antes de realizar una operación que pueda llevar a una relación con una cabecera en la que aparezcan dos atributos con el mismo nombre.

- *Operación renombrar* ρ. A partir de una relación especificada, crea una nueva copia de ésta en la que sólo se han modificado los nombres de aquellos atributos que se quieren renombrar. La sintaxis del operador es:

$$R\ \rho_{\text{atributos\_nuevos}}\ (\text{atributos\_originales})$$

  donde *R* es el nombre de la relación, *atributos_originales* son los nombres de los atributos que se quieren renombrar (separados por comas) y *atributos_nuevos* son los nuevos nombres de éstos (separados por comas).

2.3.2 Operaciones de conjuntos

Las operaciones de conjuntos son la *unión*, la *intersección*, la *diferencia* y el *producto cartesiano*.

Para poder definir estas operaciones es necesario saber lo que se entiende por *compatibilidad respecto a la unión* y *compatibilidad respecto al producto*:

- Dos relaciones son *compatibles respecto a la unión* si, y sólo si, sus cabeceras son idénticas, lo que significa que:
  - Las dos tienen el mismo conjunto de nombres de atributos.
  - Los atributos correspondientes se definen sobre el mismo dominio.
- Dos relaciones son *compatibles respecto al producto* si, y sólo si, sus cabeceras son disjuntas (no contienen nombres de atributos iguales).

Operaciones de conjuntos:

- *Unión* ∪. La unión de dos relaciones R_1 y R_2, compatibles respecto a la unión, es otra relación cuya cabecera es idéntica a la de R_1 (o a la de R_2), y cuyo cuerpo está formado por todas las tuplas pertenecientes a R_1, a R_2 o a las dos. La unión construye una relación formada por todas las tuplas que aparecen en cualquiera de las dos relaciones especificadas. La sintaxis de la operación es:

$$R_1 \cup R_2$$

- *Intersección* $\cap$. La intersección de dos relaciones R_1 y R_2, compatibles respecto a la unión, es una relación cuya cabecera es idéntica a la de R_1 (o a la de R_2) y cuyo cuerpo está formado por todas las tuplas pertenecientes tanto a R_1 como a R_2. La intersección construye una relación formada por todas las tuplas que aparecen en las dos relaciones especificadas. La sintaxis de la operación es:

$$R_1 \cap R_2$$

- *Diferencia* -. La diferencia de dos relaciones R_1 y R_2, compatibles respecto a la unión, es una relación cuya cabecera es idéntica a la de R_1 (o a la de R_2) y cuyo cuerpo está formado por todas las tuplas pertenecientes a R_1 pero no a R_2. La diferencia construye una relación formada por todas las tuplas de la primera relación que no aparecen en la segunda de las dos relaciones especificadas. La sintaxis de la operación es:

$$R_1 - R_2$$

- *Producto cartesiano* $\times$. El producto cartesiano de dos relaciones R_1 y R_2, compatibles respecto al producto, es una relación cuya cabecera es la combinación de las cabeceras de R_1 y R_2 y cuyo cuerpo está formado por el conjunto de todas las tuplas t tales que t es la combinación de una tupla t_1 perteneciente a R_1 y una tupla t_2 perteneciente a R_2. El producto cartesiano, a partir de dos relaciones específicas, construye una relación que contiene todas las combinaciones posibles de tuplas, una de cada una de las dos relaciones. La sintaxis de la operación es:

$$R_1 \times R_2$$

La unión, la intersección y el producto cartesiano son *asociativas* pero no la diferencia. Si R_1, R_2 y R_3 son relaciones arbitrarias, entonces:

$$(R_1 \cup R_2) \cup R_3 \Leftrightarrow R_1 \cup (R_2 \cup R_3) \text{ o bien } R_1 \cup R_2 \cup R_3$$

$$(R_1 \cap R_2) \cap R_3 \Leftrightarrow R_1 \cap (R_2 \cap R_3) \text{ o bien } R_1 \cap R_2 \cap R_3$$

$$(R_1 \times R_2) \times R_3 \Leftrightarrow R_1 \times (R_2 \times R_3) \text{ o bien } R_1 \times R_2 \times R_3$$

La unión, la intersección y el producto cartesiano son conmutativas pero no la diferencia. Si R_1 y R_2 son relaciones arbitrarias, entonces:

$$R_1 \cup R_2 \Leftrightarrow R_2 \cup R_1$$

$$R_1 \cap R_2 \Leftrightarrow R_2 \cap R_1$$

$$R_1 \times R_2 \Leftrightarrow R_2 \times R_1$$

El producto cartesiano de dos relaciones es conmutativo debido a que el orden de aparición de los atributos en la cabecera de la relación resultante no es relevante.

2.3.3 Operaciones relacionales

Las operaciones relacionales son la *selección*, la *proyección*, la *reunión* (natural y theta) y la *división*.

- *Selección* σ. La selección de tuplas de una relación R, es otra relación con la misma cabecera que la de R y cuyo cuerpo esta formado por las tuplas de R que verifican una *condición* impuesta a los atributos. En la *condición* pueden aparecer operadores de comparación (=, <>, >, >=) y booleanos (AND, OR, NOT). La selección extrae las tuplas especificadas de una relación dada, es decir, restringe la relación sólo a las tuplas que satisfacen la condición. Los atributos que aparecen en la *condición* deben estar definidos sobre el mismo dominio. La sintaxis de la operación es:

$$\sigma_{\text{condición}} (R)$$

- *Proyección* Π. La proyección de la relación R según los atributos A_1, A_2, ..., A_n, es otra relación que tiene como cabecera la formada por los atributos indicados y en cuyo cuerpo aparecen todas las tuplas de R restringidas a dichos atributos, eliminando las tuplas repetidas. La sintaxis de la operación es:

$$\Pi_{A_1, A_2, \ldots, A_n} (R)$$

- *Reunión*. La reunión de dos relaciones específicas, es otra relación que contiene todas las posibles combinaciones de tuplas, una de cada una de las dos relaciones, tales que las dos tuplas participantes en la combinación satisfacen una condición especificada. Se consideran dos tipos de reunión, la *reunión natural* y la *reunión theta*.

  - *Reunión natural* *. Sea (A_1, A_2, ..., A_m, B_1, B_2, ..., B_n) la cabecera de una relación R_1 y (B_1, B_2, ..., B_n, C_1, C_2, ..., C_p) la cabecera de otra relación R_2, estando los atributos del mismo nombre definidos en el mismo dominio. Si se consideran los tres atributos compuestos $A \equiv (A_1, A_2, ..., A_m)$, $B \equiv (B_1, B_2, ..., B_n)$ y $C \equiv (C_1, C_2, ..., C_p)$, la reunión natural

    de R_1 y R_2 es una relación con la cabecera (A, B, C) y un cuerpo formado por el conjunto de todas las tuplas (A:a, B:b, C:c) tales que una tupla t_1 aparece en R_1 con el valor a en A y el valor b en B, y una tupla t_2 aparece en R_2 con el valor b en B y el valor c en C. La reunión natural de dos relaciones específicas, es otra relación que contiene todas las

posibles combinaciones de tuplas, una de cada una de las dos relaciones, tales que las dos tuplas participantes en la combinación tengan los mismos valores en los atributos comunes. La sintaxis de la operación es:

$$R_1 * R_2$$

Si R_1 y R_2 no tienen atributos en común:

$$R_1 * R_2 \Leftrightarrow R_1 \times R_2$$

Por otra parte, la reunión natural es asociativa y conmutativa:

$$(R_1 * R_2) * R_3 \Leftrightarrow R_1 * (R_2 * R_3) \Leftrightarrow R_1 * R_2 * R_3$$

$$R_1 * R_2 \Leftrightarrow R_2 * R_1$$

– *Reunión theta* |×|. Sean las relaciones R_1 y R_2 compatibles respecto al producto y sea *theta* un operador. La reunión theta de la relación R_1 según el atributo A con la relación R_2 según el atributo B, es una relación con la misma cabecera que el producto cartesiano de R_1 y R_2, y un cuerpo formado por el conjunto de todas las tuplas *t*, tales que *t* pertenece a la relación resultante si la evaluación de la condición *A theta B* resulta verdadera. Los atributos A y B deberán estar definidos en el mismo dominio y la operación *theta* debe ser aplicable a dicho dominio. La reunión theta de dos relaciones específicas, es otra relación con las mismas tuplas que el producto cartesiano eliminando aquellas que no verifican la condición. Como se verá en los problemas, la reunión natural es un caso particular de la reunión theta cuando la condición es que A sea igual a B. La sintaxis de la operación es:

$$R_1 \,|\times|_{\,A\ theta\ B}\, R_2$$

Obsérvese que:

$$R_1 \,|\times|_{\,A\ theta\ B}\, R_2 \Leftrightarrow \sigma_{\,A\ theta\ B}\,(R_1 \times R_2)$$

- *División* ÷. Sea $(A_1, A_2, ..., A_m, B_1, B_2, ..., B_n)$ la cabecera de una relación R_1 y $(B_1, B_2, ..., B_n)$ la cabecera de otra relación R_2, estando los atributos del mismo nombre definidos en el mismo dominio. Si se consideran los dos atributos compuestos A y B, la división de R_1 (dividendo) entre R_2 (divisor), es otra relación con la cabecera (A) y un cuerpo formado por el conjunto de todos los valores de R_1 en el atributo A, cuyos valores correspondientes en el atributo B incluyen a todos los valores del atributo B en la relación R_2. La división toma dos relaciones, una *binaria* y otra *unaria*, y construye una

relación formada por todos los valores de un atributo de la relación binaria que concuerdan, en el otro atributo, con todos los valores en la relación unaria. La sintaxis de la operación es:

$$R_1 \div R_2$$

2.3.4 Operaciones adicionales

- *Ampliación* α. Toma una relación R y crea otra nueva relación con un atributo más que la original cuyos valores se obtienen evaluando alguna expresión de cálculo escalar. La sintaxis de la operación es:

$$R\ \alpha_{\text{cálculo\_escalar}}\ (\text{nombre\_atributo})$$

- *Resumen* Ω. Permite incorporar operaciones de agregados (cuenta, suma, promedio, máximo, mínimo, etc.). A partir de una relación R y de una lista de sus atributos, obtiene otra relación en cuya cabecera aparecen los atributos de R especificados y un nuevo atributo, con el nombre indicado, siendo los valores de este último el resultado de evaluar la expresión de agregados. La sintaxis de la operación es:

$$R\ (\text{lista\_atributos})\ \Omega_{\text{cálculo\_agregados}}\ (\text{nombre\_atributo})$$

- *División generalizada* $\div$. Dadas la relación R_1 con la cabecera (A, B) y la relación R_2 con la cabecera (B, C), donde los atributos A, B y C pueden ser compuestos, produce una relación que tiene como cabecera (A, C) y un cuerpo formado por todas las tuplas (A:a, C:c) tales que, aparece una tupla (A:a, B:b) en R_1 para todas las tuplas (B:b, C:c) que aparecen en R_2. Si C está vacío la operación se reduce a la división de R_1 entre R_2, si A está vacío la operación se reduce a la división de R_2 entre R_1, y si B está vacío la operación degenera en el producto cartesiano de R_1 y R_2. En la división generalizada se utiliza el mismo símbolo que en la división, por ser esta última un caso particular de aquella. La sintaxis de la operación es:

$$R_1 \div R_2$$

2.3.5 Asignación relacional

El objetivo de esta operación es asignar un nombre (*etiqueta*) a la relación que resulta de evaluar una expresión del álgebra relacional. La asignación es útil en consultas que requieren una expresión algebraica extensa. La sintaxis de la operación es:

$$\text{Etiqueta} \leftarrow \text{Expresión}$$

2.4 Problemas resueltos

En esta sección se proponen y resuelven problemas utilizando el álgebra relacional. Mientras no se diga lo contrario, en los problemas se hace referencia a la base de datos AUTOMÓVILES descrita en el apéndice A.

♦ Problema 2.1

Indicar la expresión que permite obtener una relación con el cuerpo y la cabecera de la relación CONCESIONARIOS, pero con los atributos *nombre* y *ciudad* denotados por *dnombre* y *dciudad*, respectivamente.

Solución:

En este problema se ilustra la forma de utilizar la operación renombrar. Al evaluarse la expresión, se obtiene una nueva relación, sin nombre propio, con el cuerpo y la cabecera de la relación CONCESIONARIOS pero con los atributos *nombre* y *ciudad* renombrados como *dnombre* y *dciudad*, respectivamente. Obsérvese que el atributo *cifc* no aparece en la expresión debido a que no se quiere modificar su nombre.

$$\text{CONCESIONARIOS } \rho_{\text{dnombre, dciudad}} \text{ (nombre, ciudad)}$$

cifc	*dnombre*	*dciudad*
0001	acar	Madrid
0002	bcar	Madrid
0003	ccar	Barcelona
0004	dcar	Valencia
0005	ecar	Bilbao

♦ Problema 2.2

Sean COCHES1 y COCHES2 dos relaciones en las que aparecen las tuplas de la relación COCHES correspondientes al modelo 'gti' y las tuplas de la relación COCHES que tienen por nombre 'ibiza', respectivamente. Indicar por qué estas dos relaciones son compatibles respecto a la unión.

COCHES1

codcoche	*nombre*	*modelo*
0002	ibiza	gti
0005	cordoba	gti
0007	megane	gti
0016	astra	gti

COCHES2

codcoche	*nombre*	*modelo*
0001	ibiza	glx
0002	ibiza	gti
0003	ibiza	gtd

Solución:

Las relaciones COCHES1 y COCHES 2 son compatibles respecto a la unión por tener el mismo conjunto de nombres de atributos y estar los atributos del mismo nombre definidos en el mismo dominio.

♦ Problema 2.3

Para las relaciones COCHES1 y COCHES2 del problema 2.2, construir la tabla que resulta al evaluar la expresión:

$$COCHES1 \cup COCHES2$$

Solución:

La unión de las dos relaciones indicadas es otra con la misma cabecera que éstas y en cuyo cuerpo están todas las tuplas de las dos relaciones. Obsérvese como no aparecen las tuplas repetidas.

codcoche	*nombre*	*modelo*
0002	ibiza	gti
0005	cordoba	gti
0007	megane	gti
0016	astra	gti
0001	ibiza	glx
0003	ibiza	gtd

♦ Problema 2.4

Para las relaciones COCHES1 y COCHES2 del problema 2.2, construir la tabla que resulta al evaluar la expresión:

$$COCHES1 \cap COCHES2$$

Solución:

La intersección de éstas relaciones es otra con la misma cabecera y en cuyo cuerpo aparecen las tuplas que están simultáneamente en ellas.

codcoche	*nombre*	*modelo*
0002	ibiza	gti

♦ Problema 2.5

Para las relaciones COCHES1 y COCHES2 del problema 2.2, construir las tablas que resultan al evaluar las expresiones:

COCHES1 - COCHES2

COCHES2 - COCHES1

Solución:

Al evaluar COCHES1 - COCHES2 se obtiene una relación con la misma cabecera que las relaciones que aparecen en la expresión y en cuyo cuerpo están las tuplas que pertenecen a COCHES1 pero no a COCHES2.

codcoche	*nombre*	*modelo*
0005	cordoba	gti
0007	megane	gti
0016	astra	gti

Al evaluar COCHES2 - COCHES1 se obtiene una relación con la misma cabecera que las relaciones que aparecen en la expresión y en cuyo cuerpo están las tuplas que pertenecen a COCHES2 pero no a COCHES1.

codcoche	*nombre*	*modelo*
0001	ibiza	glx
0003	ibiza	gtd

♦ Problema 2.6

Sean NMARCAS y NCONCESIONARIOS dos relaciones en las que aparecen los nombres de las marcas de coches y los nombres de los concesionarios, respectivamente. Indicar si éstas dos relaciones son compatibles respecto al producto.

NMARCAS

nombre
seat
renault
citroen
audi
opel
bmw

NCONCESIONARIOS

nombre
acar
bcar
ccar
dcar
ecar

Solución:

Las relaciones NMARCAS y NCONCESIONARIOS no son compatibles respecto al producto. Para que dos relaciones cumplan la condición de ser compatibles respecto al producto sus cabeceras no deben tener nombres de atributos iguales, cosa que no ocurre en este caso.

♦ Problema 2.7

Para las relaciones NMARCAS y NCONCESIONARIOS del problema 2.6, dar una expresión que permita obtener el producto cartesiano de las dos relaciones e indicar el resultado de evaluar dicha expresión.

Solución:

Para que sea posible obtener el producto cartesiano de las relaciones NMARCAS y NCONCESIONARIOS es preciso que éstas dos relaciones cumplan la condición de ser compatibles respecto al producto, lo que se consigue cambiando de nombre uno de los atributos *nombre* en cualquiera de las dos relaciones.

$$(\text{NMARCAS } \rho_{\text{mnombre}} (\text{nombre})) \times \text{NCONCESIONARIOS}$$

mnombre	*nombre*
seat	acar
seat	bcar
seat	ccar
seat	dcar
seat	ecar
renault	acar
renault	bcar
renault	ccar
renault	dcar
renault	ecar
citroen	acar
citroen	bcar
citroen	ccar
citroen	dcar
citroen	ecar

(continuación)

mnombre	*nombre*
audi	acar
audi	bcar
audi	ccar
audi	dcar
audi	ecar
opel	acar
opel	bcar
opel	ccar
opel	dcar
opel	ecar
bmw	acar
bmw	bcar
bmw	ccar
bmw	dcar
bmw	ecar

♦ Problema 2.8

Indicar una expresión que permita obtener las tuplas de la relación MARCAS para las que el atributo *ciudad* es 'Barcelona'.

Solución:

El problema se resuelve seleccionando las tuplas de la relación MARCAS que verifican la condición *ciudad* = 'Barcelona'.

$$\sigma_{\text{ciudad = 'Barcelona'}} (\text{MARCAS})$$

cifm	*nombre*	*ciudad*
0002	renault	Barcelona
0006	bmw	Barcelona

♦ Problema 2.9

Obtener las tuplas de la relación DISTRIBUCION para las que el atributo *cantidad* toma un valor mayor que 15.

Solución:

En este caso hay que seleccionar las tuplas de la relación DISTRIBUCION que verifican la condición *cantidad* > 15.

$$\sigma_{\text{cantidad > 15}} (\text{DISTRIBUCION})$$

cifc	*codcoche*	*cantidad*
0005	0016	20

♦ Problema 2.10

Obtener las tuplas de la relación CLIENTES para las que el atributo *apellido* es 'García' y el atributo *ciudad* es 'Madrid'.

Solución:

El problema se resuelve de forma similar a los dos anteriores, seleccionando las tuplas de la relación CLIENTES que cumplen la condición *apellido* = 'García' AND *ciudad* = 'Madrid'.

$$\sigma_{\text{apellido = 'García' AND ciudad = 'Madrid'}} (\text{CLIENTES})$$

Teniendo en cuenta que:

$$\sigma_{X \text{ AND } Y} (R) \Leftrightarrow (\sigma_X (R)) \cap (\sigma_Y (R))$$

otra expresión alternativa es:

$$(\sigma_{\text{apellido = 'García'}} (\text{CLIENTES})) \cap (\sigma_{\text{ciudad = 'Madrid'}} (\text{CLIENTES}))$$

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	García	Madrid
0004	María	García	Madrid

♦ Problema 2.11

Obtener las tuplas de la relación CLIENTES para las que el atributo *apellido* es 'García' o el atributo *ciudad* es 'Madrid'.

Solución:

Este problema es similar al anterior cambiando en la condición el operador AND por el operador OR.

$$\sigma_{\text{apellido = 'García' or ciudad = 'Madrid'}} (\text{CLIENTES})$$

Teniendo en cuenta que:

$$\sigma_{X \text{ OR } Y} (R) \Leftrightarrow (\sigma_X (R)) \cup (\sigma_Y (R))$$

otra expresión alternativa es:

$$(\sigma_{\text{apellido = 'García'}} (\text{CLIENTES})) \cup (\sigma_{\text{ciudad = 'Madrid'}} (\text{CLIENTES}))$$

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	García	Madrid
0003	Juan	Martín	Madrid
0004	María	García	Madrid

♦ Problema 2.12

Obtener las tuplas de la relación CLIENTES para las que el atributo *ciudad* toma un valor distinto de 'Madrid'.

Solución:

El problema se resuelve seleccionando las tuplas de la relación CLIENTES que verifican la condición NOT (*ciudad* = 'Madrid').

$$\sigma_{\text{NOT (ciudad = 'Madrid')}} (\text{CLIENTES})$$

Teniendo en cuenta que:

$$\sigma_{\text{NOT X}} (R) \Leftrightarrow R - (\sigma_{X} (R))$$

otra expresión alternativa es:

$$\text{CLIENTES} - (\sigma_{\text{ciudad = 'Madrid'}} (\text{CLIENTES}))$$

dni	*nombre*	*apellido*	*ciudad*
0002	Antonio	López	Valencia
0005	Javier	González	Barcelona
0006	Ana	López	Barcelona

♦ Problema 2.13

Obtener una relación en la que aparezcan los valores del atributo *apellido* de la relación CLIENTES.

Solución:

El problema se resuelve proyectando la relación CLIENTES según el atributo *apellido*. En la relación resultante no aparecen las tuplas repetidas.

$$\Pi_{\text{apellido}} (\text{CLIENTES})$$

apellido
García
López
Martín
González

♦ Problema 2.14

Obtener una relación en la que aparezcan los valores de los atributos *apellido* y *ciudad* de la relación CLIENTES.

Solución:

El problema es similar al anterior realizando la proyección según los atributos *apellido* y *ciudad*. En la relación resultante no aparecen las tuplas repetidas.

$$\Pi_{\text{apellido, ciudad}} (\text{CLIENTES})$$

apellido	*ciudad*
García	Madrid
López	Valencia
Martín	Madrid
González	Barcelona
López	Barcelona

♦ Problema 2.15

Obtener una relación en la que aparezcan los valores de los atributos *apellido* y *ciudad* para los clientes de 'Madrid'.

Solución:

El problema se resuelve seleccionando las tuplas de la relación CLIENTES que verifican la condición *ciudad* = 'Madrid' y proyectando la relación resultante según los atributos *apellido* y *ciudad*.

$$\Pi_{\text{apellido, ciudad}} (\sigma_{\text{ciudad = 'Madrid'}} (\text{CLIENTES}))$$

Si se aplica la asignación relacional, se puede obtener en primer lugar una relación denominada SELECCION con las tuplas de la relación CLIENTES que cumplen la condición *ciudad* = 'Madrid'.

$$\text{SELECCION} \leftarrow \sigma_{\text{ciudad = 'Madrid'}} (\text{CLIENTES})$$

y a continuación realizar la proyección de la relación SELECCION según los atributos *apellido* y *ciudad*.

$$\Pi_{\text{apellido, ciudad}} (\text{SELECCION})$$

apellido	*ciudad*
García	Madrid
Martín	Madrid

Obsérvese como en la relación resultante no aparecen las tuplas repetidas.

♦ Problema 2.16

Obtener la reunión *'mayor que'* de la relación MARCAS según el atributo *ciudad* con la relación CLIENTES según el atributo *ciudad*.

Solución:

La reunión *'mayor que'*, se puede expresar de la forma:

$$\text{RELACION1} \; |\times|_{\text{ATRIBUTO1 'mayor que' ATRIBUTO2}} \; \text{RELACION2}$$

Para que sea posible realizar esta operación, las relaciones que aparecen en ella deben ser compatibles respecto al producto, es decir, sus cabeceras no deben tener nombres de atributos iguales. Como tanto la relación MARCAS como la relación CLIENTES contienen en sus cabeceras los atributos *nombre* y *ciudad*, para poder realizar la reunión *'mayor que'*, será preciso cambiar el nombre de éstos atributos en alguna de las relaciones.

Si se aplican las operaciones *renombrar* y la *asignación relacional* como se indica en la siguiente expresión:

$$\text{NMARCAS} \leftarrow \text{MARCAS} \; \rho_{\text{mnombre, mciudad}} \; (\text{nombre, ciudad})$$

resulta la relación NMARCAS cuya cabecera (*cifm*, *mnombre*, *mciudad*) no contiene nombres de atributos iguales a los de la relación CLIENTES, y por lo tanto estas relaciones si serán compatibles respecto al producto.

Una vez obtenida la relación NMARCAS, el problema se puede enunciar como:

"Obtener la reunión *'mayor que'* de la relación NMARCAS según el atributo *mciudad* con la relación CLIENTES según el atributo *ciudad*."

La condición *'mayor que'*, que se denota mediante el operador >, aplicada a textos es de la forma:

$$\text{Texto1} > \text{Texto2}$$

Esta condición será verdadera cuando Texto1 siga en orden alfabético a Texto2 y falsa cuando Texto1 sea, en orden alfabético, anterior a Texto2 o cuando Texto1 y Texto2 sean iguales.

Por lo tanto, al evaluar:

$$\text{NMARCAS} \; |\times|_{\text{mciudad} > \text{ciudad}} \; \text{CLIENTES}$$

se obtiene una relación con la cabecera (*cifm, mnombre, mciudad, dni, nombre, apellido, ciudad*) y en cuyo cuerpo aparecen las tuplas del producto cartesiano de las relaciones NMARCAS y CLIENTES, tales que, el nombre de la ciudad correspondiente al atributo *mciudad* sigue en orden alfabético al nombre de la ciudad correspondiente al atributo *ciudad*.

En la siguiente tabla se muestran las tuplas correspondientes al producto cartesiano de las relaciones NMARCAS y CLIENTES.

cifm	*mnombre*	*mciudad*	*dni*	*nombre*	*apellido*	*ciudad*
0001	seat	Madrid	0001	Luis	Garcia	Madrid
0001	seat	Madrid	0002	Antonio	López	Valencia
0001	seat	Madrid	0003	Juan	Martín	Madrid
0001	seat	Madrid	0004	María	García	Madrid
0001	***seat***	***Madrid***	***0005***	***Javier***	***González***	***Barcelona***
0001	***seat***	***Madrid***	***0006***	***Ana***	***López***	***Barcelona***
0002	renault	Barcelona	0001	Luis	García	Madrid
0002	renault	Barcelona	0002	Antonio	López	Valencia
0002	renault	Barcelona	0003	Juan	Martín	Madrid
0002	renault	Barcelona	0004	María	García	Madrid
0002	renault	Barcelona	0005	Javier	González	Barcelona
0002	renault	Barcelona	0006	Ana	López	Barcelona
0003	***citroen***	***Valencia***	***0001***	***Luis***	***García***	***Madrid***
0003	citroen	Valencia	0002	Antonio	López	Valencia
0003	***citroen***	***Valencia***	***0003***	***Juan***	***Martín***	***Madrid***
0003	***citroen***	***Valencia***	***0004***	***María***	***García***	***Madrid***
0003	***citroen***	***Valencia***	***0005***	***Javier***	***González***	***Barcelona***
0003	***citroen***	***Valencia***	***0006***	***Ana***	***López***	***Barcelona***
0004	audi	Madrid	0001	Luis	García	Madrid
0004	audi	Madrid	0002	Antonio	López	Valencia
0004	audi	Madrid	0003	Juan	Martín	Madrid
0004	audi	Madrid	0004	María	García	Madrid
0004	***audi***	***Madrid***	***0005***	***Javier***	***González***	***Barcelona***
0004	***audi***	***Madrid***	***0006***	***Ana***	***López***	***Barcelona***
0005	opel	Bilbao	0001	Luis	García	Madrid
0005	opel	Bilbao	0002	Antonio	López	Valencia
0005	opel	Bilbao	0003	Juan	Martín	Madrid
0005	opel	Bilbao	0004	María	García	Madrid
0005	***opel***	***Bilbao***	***0005***	***Javier***	***González***	***Barcelona***
0005	***opel***	***Bilbao***	***0006***	***Ana***	***Lopez***	***Barcelona***
0006	bmw	Barcelona	0001	Luis	García	Madrid
0006	bmw	Barcelona	0002	Antonio	López	Valencia
0006	bmw	Barcelona	0003	Juan	Martín	Madrid
0006	bmw	Barcelona	0004	María	García	Madrid
0006	bmw	Barcelona	0005	Javier	González	Barcelona
0006	bmw	Barcelona	0006	Ana	López	Barcelona

donde se han destacado las tuplas de la relación resultante:

La expresión global que permite obtener la relación indicada en la siguiente tabla es:

$$(\text{MARCAS } \rho_{\text{mnombre, mciudad}} (\text{nombre, ciudad})) \; |\times|_{\text{mciudad} > \text{ciudad}} \text{CLIENTES}$$

cifm	*mnombre*	*mciudad*	*dni*	*nombre*	*apellido*	*ciudad*
0001	seat	Madrid	0005	Javier	González	Barcelona
0001	seat	Madrid	0006	Ana	López	Barcelona
0003	citroen	Valencia	0001	Luis	García	Madrid
0003	citroen	Valencia	0003	Juan	Martín	Madrid
0003	citroen	Valencia	0004	María	García	Madrid
0003	citroen	Valencia	0005	Javier	González	Barcelona
0003	citroen	Valencia	0006	Ana	López	Barcelona
0004	audi	Madrid	0005	Javier	González	Barcelona
0004	audi	Madrid	0006	Ana	López	Barcelona
0005	opel	Bilbao	0005	Javier	González	Barcelona
0005	opel	Bilbao	0006	Ana	López	Barcelona

♦ Problema 2.17

Obtener la reunión natural de las relaciones MARCAS y CLIENTES según el atributo común *ciudad*.

Solución:

Este problema se puede enunciar como:

"Obtener la reunión *'igual que'* de la relación MARCAS según el atributo *ciudad* con la relación CLIENTES según el atributo *ciudad*, de tal forma que en la relación resultante sólo aparezca un atributo *ciudad*."

La condición *'igual que'*, que se denota mediante el operador =, aplicada a textos es de la forma:

$$\text{Texto1} = \text{Texto2}$$

Esta condición será verdadera sólo cuando Texto1 y Texto2 sean iguales.

Teniendo en cuenta las consideraciones hechas en el problema anterior, la reunión natural de las relaciones MARCAS y CLIENTES según el atributo *ciudad* se obtiene mediante las siguientes operaciones:

$$\text{NMARCAS} \leftarrow \text{MARCAS } \rho_{\text{mnombre, mciudad}} (\text{nombre, ciudad})$$

$$\text{REUNION} \leftarrow \text{NMARCAS } |\times|_{\text{mciudad = ciudad}} \text{CLIENTES}$$

$$\text{PROYECCION} \leftarrow \Pi_{\text{cifm, mnombre, dni, nombre, apellido, ciudad}} (\text{CLIENTES})$$

que equivalen a la expresión:

$$(\text{MARCAS } \rho_{\text{mnombre}} (\text{NOMBRE})) * \text{CLIENTES}$$

donde se ha utilizado directamente la operación reunión natural.

En la siguiente tabla se muestran las tuplas correspondientes al producto cartesiano de las relaciones NMARCAS y CLIENTES.

En dicha tabla se han destacado las tuplas que verifican que el nombre de la ciudad correspondiente al atributo *mciudad* es igual al nombre de la ciudad correspondiente al atributo *ciudad*.

cifm	*mnombre*	*mciudad*	*dni*	*nombre*	*apellido*	*ciudad*
0001	***seat***	***Madrid***	***0001***	***Luis***	***García***	***Madrid***
0001	seat	Madrid	0002	Antonio	López	Valencia
0001	***seat***	***Madrid***	***0003***	***Juan***	***Martín***	***Madrid***
0001	***seat***	***Madrid***	***0004***	***María***	***García***	***Madrid***
0001	seat	Madrid	0005	Javier	González	Barcelona
0001	seat	Madrid	0006	Ana	López	Barcelona
0002	renault	Barcelona	0001	Luis	García	Madrid
0002	renault	Barcelona	0002	Antonio	López	Valencia
0002	renault	Barcelona	0003	Juan	Martín	Madrid
0002	renault	Barcelona	0004	María	García	Madrid
0002	***renault***	***Barcelona***	***0005***	***Javier***	***González***	***Barcelona***
0002	***renault***	***Barcelona***	***0006***	***Ana***	***López***	***Barcelona***
0003	citroen	Valencia	0001	Luis	García	Madrid
0003	***citroen***	***Valencia***	***0002***	***Antonio***	***López***	***Valencia***
0003	citroen	Valencia	0003	Juan	Martín	Madrid
0003	citroen	Valencia	0004	María	García	Madrid
0003	citroen	Valencia	0005	Javier	González	Barcelona
0003	citroen	Valencia	0006	Ana	López	Barcelona
0004	***audi***	***Madrid***	***0001***	***Luis***	***García***	***Madrid***
0004	audi	Madrid	0002	Antonio	López	Valencia
0004	***audi***	***Madrid***	***0003***	***Juan***	***Martín***	***Madrid***
0004	***audi***	***Madrid***	***0004***	***María***	***García***	***Madrid***
0004	audi	Madrid	0005	Javier	González	Barcelona
0004	audi	Madrid	0006	Ana	López	Barcelona
0005	opel	Bilbao	0001	Luis	García	Madrid
0005	opel	Bilbao	0002	Antonio	López	Valencia
0005	opel	Bilbao	0003	Juan	Martín	Madrid
0005	opel	Bilbao	0004	María	García	Madrid
0005	opel	Bilbao	0005	Javier	González	Barcelona
0005	opel	Bilbao	0006	Ana	Lopez	Barcelona
0006	bmw	Barcelona	0001	Luis	García	Madrid
0006	bmw	Barcelona	0002	Antonio	López	Valencia
0006	bmw	Barcelona	0003	Juan	Martín	Madrid
0006	bmw	Barcelona	0004	María	García	Madrid
0006	***bmw***	***Barcelona***	***0005***	***Javier***	***González***	***Barcelona***
0006	***bmw***	***Barcelona***	***0006***	***Ana***	***López***	***Barcelona***

Obsérvese como al existir en la relación MARCAS la ciudad de 'Bilbao' y no estar ésta en la relación CLIENTES, para dicha ciudad no existirá ninguna tupla en la relación resultante.

Si se seleccionan las tuplas marcadas en la tabla anterior y se elimina la columna correspondiente al atributo MCIUDAD se llega al resultado final.

cifm	*mnombre*	*dni*	*nombre*	*apellido*	*ciudad*
0001	seat	0001	Luis	García	Madrid
0001	seat	0003	Juan	Martín	Madrid
0001	seat	0004	María	García	Madrid
0002	renault	0005	Javier	González	Barcelona
0002	renault	0006	Ana	López	Barcelona
0003	citroen	0002	Antonio	López	Valencia
0004	audi	0001	Luis	García	Madrid
0004	audi	0003	Juan	Martín	Madrid
0004	audi	0004	María	García	Madrid
0006	bmw	0005	Javier	González	Barcelona
0006	bmw	0006	Ana	López	Barcelona

♦ Problema 2.18

Obtener la división entre la relación DIVIDENDO y la relación DIVISOR para los tres casos siguientes.

DIVIDENDO

atributo1	*atributo2*
dato1	dato6
dato1	dato8
dato1	dato9
dato2	dato6
dato3	dato5
dato3	dato6
dato4	dato10
dato4	dato9
dato4	dato8
dato4	dato7
dato4	dato6
dato4	dato5

DIVISOR

atributo2
dato5

DIVISOR

atributo2
dato6
dato8

DIVISOR

atributo2
dato5
dato6
dato7
dato8
dato9
dato10

Solución:

En los tres casos la relación resultante tendrá como cabecera *atributo1* y en ella aparecerán todos los valores de dicho atributo de la relación DIVIDENDO

que concuerden, en los correspondientes valores de *atributo2*, con todos los valores de la relación DIVISOR.

atributo1
dato3
dato4

atributo1
dato1
dato4

atributo1
dato4

♦ Problema 2.19

Ampliar la relación DISTRIBUCION con un nuevo atributo en el que aparezca el resultado de multiplicar por dos el valor que toma en cada tupla el atributo *cantidad*.

Solución:

La siguiente expresión, donde se utiliza la operación *ampliación*, permite obtener la relación especificada en el enunciado, siendo *cantidad2* el nuevo atributo añadido a la cabecera de la relación DISTRIBUCION.

$$\text{DISTRIBUCION } \alpha_{\text{cantidad} * 2} (\text{cantidad2})$$

cifc	*codcoche*	*cantidad*	*cantidad2*
0001	0001	3	6
0001	0005	7	14
0001	0006	7	14
0002	0006	5	10
0002	0008	10	20
0002	0009	10	20
0003	0010	5	10
0003	0011	3	6
0003	0012	5	10
0004	0013	10	20
0004	0014	5	10
0005	0015	10	20
0005	0016	20	40
0005	0017	8	16

♦ Problema 2.20

Ampliar la relación DISTRIBUCION con un nuevo atributo, en el que aparezca el resultado de multiplicar por dos el valor que toma en cada tupla el atributo *cantidad*, y obtener las tuplas de la relación resultante en las que el nuevo atributo toma un valor menor que 10.

Solución:

Este problema se diferencia del anterior en que, además de ampliar la relación DISTRIBUCION, se debe realizar la selección de tuplas que verifican la condición *cantidad2* < 10.

$$\sigma_{\text{cantidad2} < 10} (\text{DISTRIBUCION } \alpha_{\text{cantidad} * 2} (\text{cantidad2}))$$

cifc	*codcoche*	*cantidad*	*cantidad2*
0001	0001	3	6
0003	0011	3	6

♦ Problema 2.21

Obtener una relación, con un único atributo, en la que aparezca la suma de todos los valores del atributo *cantidad* de la relación DISTRIBUCION.

Solución:

Para calcular la suma de los valores indicados se utiliza la operación *resumen* con el operador de agregados *SUM*. Como en este caso no se quiere añadir el nuevo atributo a ninguna lista de atributos, se deja en blanco el espacio comprendido entre los paréntesis que siguen al nombre de la relación. El nombre asignado al atributo correspondiente a la suma de valores es *scantidad*.

$$\text{DISTRIBUCION } (\,) \, \Omega_{\text{SUM (cantidad)}} (\text{scantidad})$$

scantidad
108

♦ Problema 2.22

Obtener una relación en la que aparezcan los distintos valores que toma el atributo *cifc* en la relación DISTRIBUCION junto con la suma de los valores del atributo *cantidad* correspondientes a cada valor del atributo *cifc*.

Solución:

La expresión que permite determinar la relación especificada es similar a la del problema anterior.

El atributo por el que se agrupan las sumas se sitúa entre los paréntesis que siguen al nombre de la relación. El nombre del atributo correspondiente a la suma de valores del atributo *cantidad* para cada valor del atributo *cifc* es *scantidad*.

$$\text{DISTRIBUCION (cifc) } \Omega_{\text{SUM (cantidad)}} \text{ (scantidad)}$$

cifc	*scantidad*
0001	17
0002	25
0003	13
0004	15
0005	38

♦ Problema 2.23

Obtener el resultado de dividir la relación MARCO entre la relación que resulta al seleccionar las tuplas de la relación COCHES con el atributo *nombre* igual a 'ibiza', proyectada según los atributos *codcoche* y *nombre*.

Solución:

En este es caso hay que realizar la *división generalizada*:

$$\text{MARCO} \div \text{NCOCHES}$$

donde la relación dividendo tiene como cabecera (cifm, codcoche) y la cabecera de la relación divisor es (codcoche, nombre).

La relación NCOCHES se obtiene a partir de la siguiente expresión en la que se ha utilizado la asignación relacional:

$$\text{NCOCHES} \leftarrow \Pi_{\text{codcoche, nombre}} (\sigma_{\text{nombre = 'ibiza'}} (\text{COCHES}))$$

Por lo tanto, la expresión global será:

$$\text{MARCO} \div (\Pi_{\text{codcoche, nombre}} (\sigma_{\text{nombre = 'ibiza'}} (\text{COCHES})))$$

cifm	*nombre*
0001	ibiza

♦ **Problema 2.24**

Obtener los nombres de las marcas que tienen modelos 'gtd'.

Solución:

Como los nombres de las marcas se encuentran en la relación MARCAS y los modelos de los coches se encuentran en la relación COCHES, estando sus tuplas relacionadas como se indica en la relación MARCO, para resolver el problema será preciso utilizar estas tres relaciones.

Si se plantea como primer objetivo el obtener una relación en la que aparezcan los nombres de las marcas y los modelos de los coches, esto se puede conseguir a partir de la reunión natural de las relaciones MARCAS, MARCO y COCHES, siendo *cifm* el atributo común de MARCAS y MARCO, y *codcoche* el atributo común de MARCO y COCHES.

Para que en la relación resultante no existan dos atributos con el mismo nombre, como paso previo, se puede renombrar el atributo *nombre* de la relación COCHES mediante la siguiente expresión:

$$NCOCHES \leftarrow COCHES\ \rho_{cnombre}\ (nombre)$$

Una vez realizada esta operación, se lleva a cabo la reunión natural:

$$REUNION \leftarrow MARCAS * MARCO * NCOCHES$$

con la que se consigue tener en una misma relación los nombres de las marcas y los modelos de los coches.

Por otra parte, al estarse interesado sólo en los modelos 'gtd', si se realiza la selección:

$$SELECCION \leftarrow \sigma_{modelo = 'gtd'}\ (REUNION)$$

se restringe la relación REUNION a las tuplas de los modelos 'gtd'.

Para finalizar, si se proyecta la selección anterior según el atributo *nombre*:

$$\Pi_{nombre}\ (SELECCION)$$

se consigue la relación con los nombres de las marcas con modelos 'gtd'.

La forma de resolver el problema no es única. Así, mediante las operaciones:

$NCOCHES \leftarrow COCHES\ \rho_{cnombre}\ (nombre)$

SELECCION ← $\sigma_{\text{modelo = 'gtd'}}$ (NCOCHES)

REUNION1 ← SELECCION * MARCO

REUNION2 ← REUNION1 * MARCAS

Se llega a la relación REUNION2 indicada en la siguiente tabla:

REUNION2

codcoche	*cnombre*	*modelo*	*cifm*	*nombre*	*Ciudad*
0003	ibiza	gtd	0001	seat	Madrid
0004	toledo	gtd	0001	seat	Madrid
0008	laguna	gtd	0002	renault	Barcelona
0012	xantia	gtd	0003	citroen	Valencia

donde las tres primeras columnas corresponden a la relación SELECCION y las cuatro primeras columnas a la relación REUNION1.

Por último, mediante la proyección:

$$\Pi_{\text{nombre}} \text{ (REUNION2)}$$

se consigue, como en el caso anterior, una relación con los nombres de las marcas que tienen modelos 'gtd'.

nombre
seat
renault
citroen

♦ Problema 2.25

Obtener el nombre de las marcas de las que se han vendido coches de color 'rojo'.

Solución:

Para resolver el problema se puede comenzar seleccionando las tuplas de la relación VENTAS para las que el atributo *color* toma el valor 'rojo' y, a continuación, proyectar la relación resultante según el atributo *codcoche*. Con ello se consigue una relación, denotada por NVENTAS, en la que sólo aparece el atributo *codcoche* para los coches de color 'rojo'.

Realizando la reunión natural de las relaciones NVENTAS y MARCO, según el atributo común *codcoche*, y proyectando la relación resultante por el atributo *cifm*, se obtienen los valores de este atributo correspondientes a las marcas de las que se han vendido coches de color 'rojo'. Por último, para conocer los nombres de las marcas especificadas en el enunciado, se realiza la reunión natural de la relación anterior con la relación MARCAS, según el atributo común *cifm*, y se proyecta por el atributo *nombre*.

$$NVENTAS \leftarrow \Pi_{codcoche}(\sigma_{color} = \text{'rojo'}(VENTAS))$$

$$\Pi_{nombre}(\Pi_{cifm}(NVENTAS * MARCO) * MARCAS)$$

nombre
seat
renault
citroen

♦ Problema 2.26

Sea NCOCHES una relación con la misma cabecera que COCHES y en cuyo cuerpo aparecen las tuplas de esta última relación para las que el atributo *modelo* toma el valor 'gti' o 'gtd'.

NCOCHES

codcoche	*nombre*	*modelo*
0002	Ibiza	gti
0003	ibiza	gtd
0004	toledo	gtd
0005	cordoba	gti
0007	megane	gti
0008	laguna	gtd
0012	xantia	gtd
0016	astra	gti

Utilizando las relaciones NCOCHES y MARCO, obtener mediante la operación *división* los valores del atributo *cifm* para las marcas que dispongan de los modelos 'gtd' y 'gti'.

Solución:

Para resolver el problema se tiene que disponer dos relaciones DIVIDENDO y DIVISOR con cabeceras (*cifm*, *modelo*) y (*modelo*), respectivamente.

Si se proyecta la relación NCOCHES según el atributo *modelo*, en la relación resultante (DIVISOR) aparecen dos tuplas con los valores 'gti' y 'gtd', respectivamente.

Por otra parte, si se realiza la reunión natural de las relaciones MARCO y NCOCHES, según el atributo común *codcoche*, y se proyecta por los atributos *cifm* y *modelo*, se obtiene una relación (DIVIDENDO) con los valores de *cifm* para las marcas que disponen de modelos 'gti' y/o 'gtd'.

cifm	*modelo*
0001	gti
0001	gtd
0002	gti
0002	gtd
0003	gtd
0005	gti

Por último, mediante la operación división, aplicada a estas dos relaciones (DIVIDENDO ÷ DIVISOR), se obtienen los valores de *cifm* indicados en el enunciado.

$$\Pi_{\text{cifm, modelo}} (\text{MARCO} * \text{NCOCHES}) \div \Pi_{\text{modelo}} (\text{NCOCHES})$$

cifm
0001
0002

♦ Problema 2.27

Obtener el nombre de los coches que tengan al menos los mismos modelos que el coche de nombre 'cordoba'.

Solución:

El problema se resuelve mediante la división de las relaciones DIVIDENDO entre DIVISOR, siendo DIVIDENDO la relación resultante de proyectar la relación COCHES, según los atributos *nombre* y *modelo*, y DIVISOR la relación que resulta al proyectar, según el atributo *modelo*, las tuplas de la relación COCHES para las que el atributo *nombre* es igual a 'cordoba'.

$$\text{DIVIDENDO} \leftarrow \Pi_{\text{nombre, modelo}} (\text{COCHES})$$

$$\text{DIVISOR} \leftarrow \Pi_{\text{modelo}} (\sigma_{\text{nombre = 'cordoba'}} (\text{COCHES}))$$

DIVIDENDO ÷ DIVISOR

nombre
ibiza
cordoba
megane
astra

♦ Problema 2.28

Obtener los nombres de los coches que no tengan modelo 'gtd'.

Solución:

El problema se puede resolver eliminado de la relación que contiene todos los nombres de coches, los nombres de los coches que tienen modelo 'gtd'.

$$\Pi_{\text{nombre}} (\text{COCHES}) - \Pi_{\text{nombre}} (\sigma_{\text{modelo = 'gtd'}} (\text{COCHES}))$$

nombre
cordoba
megane
zx
a4
astra
corsa
300
500
700

♦ Problema 2.29

Del conjunto de las ocho operaciones básicos del álgebra relacional, la *unión*, *diferencia*, *producto cartesiano*, *selección* y *proyección* se pueden considerar como primitivas. Definir la *reunión*, *intersección* y *división* en términos de las operaciones primitivas.

Solución:

– Reunión theta:

$$R_1 |\times|_{A \text{ theta } B} R_2 \Leftrightarrow \sigma_{A \text{ theta } B} (R_1 \times R_2)$$

- Reunión natural (B' es el atributo de R_2 con el mismo nombre que el atributo B de R_1 renombrado para poder aplicar el operador ×):

$$R_1 * R_2 \Leftrightarrow \Pi_{A, B, C} (R_1 |\times|_{B = B'} R_2) \Leftrightarrow \Pi_{A, B, C} (\sigma_{B = B'} (R_1 \times R_2))$$

- Intersección:

$$R_1 \cap R_2 \Leftrightarrow R_1 - (R_1 - R_2) \Leftrightarrow R_2 - (R_2 - R_1)$$

- División (A y B son los atributos de R_1 y B es el atributo de R2):

$$R_1 \div R_2 = \Pi_A (R_1) - \Pi_A ((\Pi_A (R_1) \times R_2) - R_1)$$

♦ Problema 2.30

Obtener todas las tuplas de la relación CONCESIONARIOS.

Solución

$$\Pi_{cifc, nombre, ciudad} (CONCESIONARIOS)$$

cifc	*nombre*	*ciudad*
0001	acar	Madrid
0002	bcar	Madrid
0003	ccar	Barcelona
0004	dcar	Valencia
0005	ecar	Bilbao

♦ Problema 2.31

Obtener todas las tuplas de la relación CLIENTES correspondientes a clientes de 'Madrid'.

Solución:

$$\sigma_{ciudad = 'Madrid'} (CLIENTES)$$

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	García	Madrid
0003	Juan	Martín	Madrid
0004	María	García	Madrid

♦ Problema 2.32

Obtener el *cifc* de todos los concesionarios que disponen de una cantidad de coches mayor que 18 unidades.

Solución:

Este problema se resuelve utilizando la relación DISTRIBUCION. En dicha relación se observa como para un mismo *cifc* pueden aparecer más de un valor en el atributo *cantidad*. Por lo tanto, como primer paso hay que obtener una relación SUMA en la que aparezcan los distintos valores que toma el atributo *cifc*, en la relación DISTRIBUCION, junto con la suma de los valores del atributo *cantidad* para cada valor de *cifc* (ver problema 2.22).

$$SUMA \leftarrow DISTRIBUCION\ (cifc)\ \Omega_{SUM\ (cantidad)}\ (scantidad)$$

SUMA

cifc	*scantidad*
0001	17
0002	25
0003	13
0004	15
0005	38

Una vez determinada la relación SUMA se llega al resultado final mediante la expresión:

$$\Pi_{cifc}\ (\sigma_{scantidad > 18}\ (SUMA))$$

cifc
0002
0005

♦ Problema 2.33

Obtener el *cifc* de todos los concesionarios que disponen de una cantidad de coches comprendida entre 10 y 18 unidades, ambas inclusive.

Solución:

Este problema se resuelve de forma similar al anterior:

$$SUMA \leftarrow DISTRIBUCION\ (cifc)\ \Omega_{SUM\ (cantidad)}\ (scantidad)$$

$$\Pi_{\text{cifc}} \left(\sigma_{\text{scantidad} >= 10 \text{ AND } \text{scantidad} <= 18} (\text{SUMA})\right)$$

Otra expresión alternativa es:

$$\Pi_{\text{cifc}} \left(\sigma_{\text{scantidad} >= 10} (\text{SUMA}) \cap \sigma_{\text{scantidad} <= 18} (\text{SUMA})\right)$$

cifc
0001
0003
0004

♦ Problema 2.34

Obtener el *cifc* de todos los concesionarios que disponen de una cantidad de coches mayor que 15 o menor que 5 unidades.

Solución:

Como en los dos problemas anteriores, primero se determina la relación suma:

$$\text{SUMA} \leftarrow \text{DISTRIBUCION (cifc)}\ \Omega_{\text{SUM (cantidad)}}\ (\text{scantidad})$$

A partir de esta última relación, el problema se resuelve con la expresión:

$$\Pi_{\text{cifc}} \left(\sigma_{\text{scantidad} > 15 \text{ OR } \text{scantidad} < 5} (\text{SUMA})\right)$$

Otra expresión alternativa es:

$$\Pi_{\text{cifc}} \left(\sigma_{\text{scantidad} > 15} (\text{SUMA}) \cup \sigma_{\text{scantidad} < 5} (\text{SUMA})\right)$$

cifc
0001
0002
0005
...

♦ Problema 2.35

Obtener todas las parejas de valores de los atributos *cifm* de la relación MARCAS y *dni* de la relación CLIENTES que sean de la misma ciudad.

Solución:

Para resolver este problema, primero se realiza la reunión natural de las relaciones MARCAS y CLIENTES, como se indicó en el problema 2.17, y a continuación se proyecta la relación resultante según los atributos *cifm* y *dni*.

$$\Pi_{\text{cifm, dni}} (\text{MARCAS } \rho_{\text{mnombre}} (\text{nombre}) * \text{CLIENTES})$$

cifm	*dni*
0001	0001
0001	0003
0001	0004
0002	0005
0002	0006
0003	0002
0004	0001
0004	0003
0004	0004
0006	0005
0006	0006

♦ **Problema 2.36**

Obtener todas las parejas de valores de los atributos *cifm* de la relación MARCAS y *dni* de la relación CLIENTES que *no* sean de la misma ciudad.

Solución:

El problema se soluciona realizando la reunión *'distinto que'* de las relaciones MARCAS y CLIENTES y proyectando la relación resultante por los atributos *cifm* y *dni*.

La reunión *'distinto que'* se lleva a cabo de forma similar a la indicada en el problema 2.16 para la reunión *'mayor que'*.

En el presente problema, en lugar de renombrar uno de los atributos *nombre* (como se hizo en el problema 2.16) se han proyectado las relaciones según los atributos que interesan para realizar la reunión.

$$\text{NMARCAS} \leftarrow \Pi_{\text{cifm, mciudad}} (\text{MARCAS } \rho_{\text{mciudad}} (\text{CIUDAD}))$$

$$\text{NCLIENTES} \leftarrow \Pi_{\text{dni, ciudad}} (\text{CLIENTES})$$

$$\Pi_{\text{cifm, dni}} (\text{NMARCAS } |\times|_{\text{mciudad} <> \text{ciudad}} \text{NCLIENTES})$$

cifm	*dni*
0001	0002
0001	0005
0001	0006
0002	0001
0002	0002
0002	0003
0002	0004
0003	0001
0003	0003
0003	0004
0003	0005
0003	0006
0004	0002

(continuación)

cifm	*dni*
0004	0005
0004	0006
0005	0001
0005	0002
0005	0003
0005	0004
0005	0005
0005	0006
0006	0001
0006	0002
0006	0003
0006	0004

♦ Problema 2.37

Obtener los valores del atributo *codcoche* para los coches que se encuentran en algún concesionario de 'Barcelona'.

Solución:

Una forma de resolver este problema es realizando la reunión natural de las relaciones CONCESIONARIOS y DISTRIBUCION según el atributo común *cifc* y, una vez seleccionadas las tuplas que verifican la condición *ciudad* = 'Barcelona', proyectar la relación resultante por el atributo *codcoche*.

$\Pi_{\text{codcoche}} (\sigma_{\text{ciudad = 'Barcelona'}} (\text{CONCESIONARIOS} * \text{DISTRIBUCION}))$

Otra alternativa es realizar la reunión natural de la relación que resulta de, seleccionar las tuplas que verifican la condición *ciudad* = 'Barcelona' en la relación CONCESIONARIOS, con la relación DISTRIBUCION y proyectar la relación resultante según el atributo *codcoche*.

$\Pi_{\text{codcoche}} (\sigma_{\text{ciudad = 'Barcelona'}} (\text{CONCESIONARIOS}) * \text{DISTRIBUCION})$

Obsérvese la diferencia existente entre las relaciones que resultan de la reunión natural con una u otra expresión.

codcoche
0010
0011
0012

♦ Problema 2.38

Obtener el valor del atributo *codcoche* de aquellos coches vendidos a clientes de 'Madrid'.

Solución:

Este problema se puede resolver de forma similar al anterior, a partir de la reunión natural de las relaciones CLIENTES y VENTAS según el atributo común *dni*.

$$\Pi_{\text{codcoche}} (\sigma_{\text{ciudad = 'Madrid'}} (\text{CLIENTES} * \text{VENTAS}))$$

Otra expresión alternativa es:

$$\Pi_{\text{codcoche}} (\sigma_{\text{ciudad = 'Madrid'}} (\text{CLIENTES}) * \text{VENTAS})$$

codcoche
0001
0006
0011
0008

♦ Problema 2.39

Obtener los valores del atributo *codcoche* para los coches que han sido adquiridos por un cliente de 'Madrid' en un concesionario de 'Madrid'.

Solución:

Para resolver el problema es necesario utilizar las relaciones CLIENTES, VENTAS y CONCESIONARIOS. El proceso de resolución consiste en encontrar los valores de los atributos *dni*, para los clientes de 'Madrid', y *cifc*, para los concesionarios de 'Madrid', y obtener los valores del atributo *codcoche* que en la relación VENTAS tienen en su tupla un valor de *cifc* y un valor de *dni* de los obtenidos anteriormente.

Para ello se pueden realizar las siguientes operaciones:

$$\text{SELECCION1} \leftarrow \sigma_{\text{ciudad = 'Madrid'}} (\text{CLIENTES})$$

$$\text{PROYECCION1} \leftarrow \Pi_{\text{dni}} (\text{SELECCION1})$$

$$\text{SELECCION2} \leftarrow \sigma_{\text{ciudad = 'Madrid'}} (\text{CONCESIONARIOS})$$

PROYECCION2 ← Π_{dni} (SELECCION2)

REUNION1 ← PROYECCION1 * VENTAS

REUNION2 ← REUNION1 * PROYECCION2

$\Pi_{codcoche}$ (REUNION2)

Los atributos comunes utilizados en las reuniones naturales son *dni* para REUNION1 y *cifc* para REUNION2.

Otra forma alternativa, para resolver el problema, es:

NCLIENTES ← CLIENTES $\rho_{cnombre, cciudad}$ (nombre, ciudad)

REUNION ← NCLIENTES * VENTAS * CONCESIONARIOS

SELECCION ← $\sigma_{cciudad = 'Madrid' AND ciudad = 'Madrid'}$ (REUNION)

$\Pi_{codcoche}$ (SELECCION)

Como en el caso anterior, los atributos comunes utilizados en las reuniones naturales son *dni* y *cifc*.

codcoche
0001
0008
0006

♦ Problema 2.40

Obtener los valores del atributo *codcoche* para los coches comprados en un concesionario de la misma ciudad que la del cliente que lo compra.

Solución:

Como en el problema anterior, es necesario utilizar las relaciones CLIENTES, VENTAS y CONCESIONARIOS. El problema se resuelve de forma sencilla a partir de la reunión natural de estas tablas, renombrando previamente uno de los atributos *nombre*, y proyectando la relación resultante según el atributo *codcoche*.

NCLIENTES ← CLIENTES $\rho_{cnombre}$ (nombre)

$\Pi_{codcoche}$ (NCLIENTES * VENTAS * CONCESIONARIOS)

Obsérvese que la reunión natural NCLIENTES * VENTAS se realiza por el atributo común *dni*, mientras que la reunión natural de esta última relación con la relación CONCESIONARIOS se realiza mediante el atributo compuesto (*cifc*, *ciudad*). Esto obliga a que en la relación resultante aparezcan sólo las tuplas relativas a los coches comprados en un concesionario de la misma ciudad que la del cliente que lo compra.

codcoche
0001
0008
0006

♦ Problema 2.41

Obtener los valores del atributo *codcoche* para los coches comprados en un concesionario de distinta ciudad que la del cliente que lo compra.

Solución:

La forma de resolver este problema es similar a la indicada como alternativa en el problema 2.39, imponiendo como condición que la ciudad del cliente sea distinta de la ciudad del concesionario.

$$NCLIENTES \leftarrow CLIENTES\ \rho_{cnombre,\ cciudad}\ (nombre,\ ciudad)$$

$$REUNION \leftarrow NCLIENTES * VENTAS * CONCESIONARIOS$$

$$SELECCION \leftarrow \sigma_{cciudad <> ciudad}\ (REUNION)$$

$$\Pi_{codcoche}\ (SELECCION)$$

codcoche
0005
0011
0014

♦ Problema 2.42

Obtener todas las parejas de nombres de marcas que sean de la misma ciudad.

Solución:

El proceso de resolución de este problema consiste en realizar la reunión natural de la relación MARCAS consigo misma, según el atributo *ciudad*, y proyectar la relación resultante sobre los campos nombre. El propósito de la condición *nombre* < *mnombre* es eliminar las parejas (nombre1, nombre1) de nombres iguales y garantizar la aparición de una sola de las parejas (nombre1, nombre2) y (nombre2, nombre1).

$NMARCAS \leftarrow MARCAS\ \rho_{mnombre}\ (nombre)$

$PROYEC1 \leftarrow \Pi_{nombre,\ ciudad}\ (MARCAS)$

$PROYEC2 \leftarrow \Pi_{mnombre,\ ciudad}\ (NMARCAS)$

$\Pi_{nombre,\ mnombre}\ (\sigma_{nombre\ <\ mnombre}\ (PROYEC1 * PROYEC2))$

nombre	*mnombre*
audi	seat
bmw	renault

♦ Problema 2.43

Obtener todas las parejas de modelos de coches que tengan el mismo nombre de coche y su marca sea de la ciudad de 'Bilbao'.

Solución:

Este problema se resuelve de forma similar al problema anterior utilizando las relaciones COCHES, MARCO y MARCAS.

El primer paso consiste en reunir la tabla COCHES consigo misma, según el atributo *nombre*. Con ello se consiguen las tuplas en las que aparecen las parejas de modelos de coches que tienen el mismo nombre de coche. El propósito de la condición *modelo* < *nmodelo* es eliminar las parejas (modelo1, modelo1) de modelos iguales y garantizar la aparición de una sola de las parejas (modelo1, modelo2) y (modelo2, modelo1).

$NCOCHES \leftarrow COCHES\ \rho_{nmodelo}\ (modelo)$

$PROYECCION \leftarrow \Pi_{nombre,\ nmodelo}\ (NCOCHES)$

$SELECCION \leftarrow \sigma_{modelo\ <\ nmodelo}\ (COCHES * PROYECCION)$

A continuación se realiza la reunión natural de las relaciones SELECCION y MARCO, según el atributo *codcoche*, para conseguir los valores del atributo *cifm* de la marca de cada pareja de modelos de coche.

$$\text{REUNION1} \leftarrow \Pi_{\text{modelo, nmodelo, cifm}} (\text{SELECCION} * \text{MARCO})$$

La siguiente operación es la reunión natural de las relaciones REUNION1 y MARCAS, según el atributo *cifm*, con lo que se consigue asociar a cada valor del atributo *cifm* la ciudad de su marca.

$$\text{REUNION2} \leftarrow \text{REUNION1} * \text{MARCAS}$$

Por último, se seleccionan las tuplas cuya ciudad de marca sea igual a 'Bilbao' y se proyecta la relación resultante sobre los campos modelo.

$$\Pi_{\text{modelo, nmodelo}} (\sigma_{\text{ciudad = 'Bilbao'}} (\text{REUNION2}))$$

modelo	*nmodelo*
caravan	gti

♦ Problema 2.44

Obtener el número total de nombres de las marcas de 'Madrid'.

Solución:

Para resolver este problema primero se seleccionan las tuplas de la relación MARCAS que verifican la condición *ciudad* = 'Madrid' y a continuación se aplica la operación *resumen* con el operador de agregados 'COUNT'. Como no se quiere añadir el nuevo atributo *cnombre* a ninguna lista de atributos, se deja en blanco el espacio comprendido entre los paréntesis que siguen al nombre de la relación.

$$(\sigma_{\text{ciudad = 'Madrid'}} (\text{MARCAS}))\ (\ )\ \Omega_{\text{COUNT (nombre)}} (\text{cnombre})$$

cnombre
2

♦ Problema 2.45

Obtener la media de todos los valores del atributo *cantidad* de la relación DISTRIBUCION.

Solución:

$$\text{DISTRIBUCION}\ (\ )\ \Omega_{\text{AVG (cantidad)}}\ (\text{mcantidad})$$

mcantidad
7.7

♦ Problema 2.46

Obtener el máximo valor que toma el atributo *dni* para los clientes de 'Madrid'.

Solución:

$$(\sigma_{\text{ciudad = 'Madrid'}}\ (\text{CLIENTES}))\ (\ )\ \Omega_{\text{MAX (dni)}}\ (\text{maxdni})$$

maxdni
0004

♦ Problema 2.47

Obtener el mínimo valor que toma el atributo *dni* para los clientes que han comprado un coche de color 'blanco'.

Solución:

$$(\sigma_{\text{color = 'blanco'}}\ (\text{VENTAS}))\ (\ )\ \Omega_{\text{MIN (dni)}}\ (\text{mindni})$$

mindni
0001

♦ Problema 2.48

Obtener el *dni* de los clientes que han comprado algún coche en un concesionario de 'Madrid'.

Solución:

$$\Pi_{dni}(\sigma_{ciudad = 'Madrid'}(CONCESIONARIOS) * VENTAS)$$

dni
0001
0002
0003

♦ Problema 2.49

Obtener los colores de los coches vendidos por el concesionario 'acar'.

Solución

$$\Pi_{color}(\sigma_{nombre = 'acar'}(CONCESIONARIOS) * VENTAS)$$

color
blanco
rojo

♦ Problema 2.50

Obtener los valores del atributo *codcoche* para los coches vendidos por algún concesionario de 'Madrid'.

Solución:

$$\Pi_{codcoche}((\sigma_{ciudad = 'Madrid'}(CONCESIONARIOS)) * VENTAS)$$

codcoche
001
005
008
006

♦ Problema 2.51

Obtener el nombre y el modelo de los coches vendidos por algún concesionario de 'Barcelona'.

Solución:

$$SELECCION \leftarrow \sigma_{ciudad = 'Barcelona'} (CONCESIONARIOS)$$

$$PROY\_REUNION \leftarrow \Pi_{codcoche} (SELECCION * VENTAS)$$

$$\Pi_{nombre, modelo} (PROY\_REUNION * COCHES)$$

nombre	*modelo*
zx	td

♦ Problema 2.52

Obtener todos los nombres de los clientes que hayan adquirido algún coche en el concesionario 'dcar'.

Solución:

$$PROY\_SELEC \leftarrow \Pi_{cifc} (\sigma_{nombre = 'dcar'} (CONCESIONARIOS))$$

$$PROY\_REUNION \leftarrow \Pi_{nombre, cifc} (CLIENTES * VENTAS)$$

$$\Pi_{nombre} (PROY\_SELEC * PROY\_REUNION)$$

nombre
Javier

♦ Problema 2.53

Obtener el nombre y el apellido de los clientes que han adquirido un coche modelo 'gti' de color 'blanco'.

Solución:

$$SELECCION1 \leftarrow \sigma_{modelo = 'gti'} (COCHES\ \rho_{cnombre} (nombre))$$

$$SELECCION2 \leftarrow \sigma_{color = 'blanco'} (VENTAS)$$
$$REUNION \leftarrow CLIENTES * SELECCION2 * SELECCION1$$

$$\Pi_{nombre, apellido} (REUNION)$$

nombre	*apellido*

♦ **Problema 2.54**

Obtener el nombre y el apellido de los clientes que han adquirido un automóvil en un concesionario que dispone de coches del modelo 'gti'.

Solución:

SELECCION ← $\sigma_{\text{modelo = 'gti'}}$ (COCHES)

PROY_REUNION ← Π_{cifc} (DISTRIBUCION * SELECCION)

REUNION ← CLIENTES * VENTAS * PROY_REUNION

$\Pi_{\text{nombre, apellido}}$ (REUNION)

nombre	*apellido*
Luis	García
Antonio	López

♦ **Problema 2.55**

Obtener el nombre y el apellido de los clientes que han adquirido algún coche en un concesionario de 'Madrid' que dispone de coches del modelo 'gti'.

Solución:

PROY_SELEC1 ← Π_{codcoche} ($\sigma_{\text{modelo = 'gti'}}$ (COCHES))

PROY_SELEC2 ← Π_{cifc} ($\sigma_{\text{ciudad = 'Madrid'}}$ (CONCESIONARIOS))

REUNION1 ← CLIENTES * VENTAS

REUNION2 ← REUNION1 * PROY_SELEC1

REUNION3 ← REUNION2 * DISTRIBUCION

REUNION4 ← REUNION3 * PROY_SELEC2

$\Pi_{\text{nombre, apellido}}$ (REUNION4)

nombre	*apellido*
Luis	García

♦ Problema 2.56

Obtener el nombre y el apellido de los clientes cuyo *dni* es menor que el correspondiente a alguno de los clientes de nombre 'Juan' y apellido 'Martín'.

Solución:

SELECCION ← $\sigma_{\text{nombre = 'Juan' AND apellido = 'Martin'}}$ (CLIENTES)

PROYECCION ← Π_{dni} (SELECCION)

RENOMBRAR ← PROYECCION ρ_{ndni} (dni)

$\Pi_{\text{nombre, apellido}}$ (CLIENTES $|\times|_{\text{dni < ndni}}$ RENOMBRAR)

nombre	*apellido*
Luis	García
Antonio	López

♦ Problema 2.57

Obtener el nombre y el apellido de los clientes cuyo *dni* es menor que el de todos los clientes que son de 'Barcelona'.

Solución:

SELECCION ← $\sigma_{\text{ciudad = 'Barcelona'}}$ (CLIENTES)

MINIMO ← SELECCION ( ) $\Omega_{\text{MIN (dni)}}$ (mindni)

$\Pi_{\text{nombre, apellido}}$ (CLIENTES $|\times|_{\text{dni < mindni}}$ MINIMO)

nombre	*apellido*
Luis	García
Antonio	López
Juan	Martín
María	García

♦ Problema 2.58

Obtener el nombre y el apellido de los clientes que han comprado como mínimo un coche 'blanco' y un coche 'rojo'.

Solución:

$$VENTA1 \leftarrow \Pi_{dni} (\sigma_{color = 'blanco'} (VENTAS))$$

$$VENTA2 \leftarrow \Pi_{ndni} ((\sigma_{color = 'rojo'} (VENTAS)) \rho_{ndni} (dni))$$

$$\Pi_{nombre, apellido} (VENTA1 \; |\times|_{dni = ndni} \; VENTA2 * CLIENTES))$$

nombre	*apellido*
Luis	García

♦ Problema 2.59

Obtener el *dni* de los clientes cuya ciudad sea la última de la lista alfabética de las ciudades donde hay concesionarios.

Solución:

$$ULTIMA \leftarrow CONCESIONARIOS ( ) \Omega_{MAX (ciudad)} (ciudad)$$

$$\Pi_{dni} (ULTIMA * CLIENTES)$$

dni
0002

♦ Problema 2.60

Obtener los valores del atributo *dni* para los clientes que sólo han comprado coches al concesionario con *cifc* igual a '0001'.

Solución:

Este problema se puede plantear como: "obtener los valores del atributo *dni* para los clientes que en la relación VENTAS no les corresponde ningún valor de *cifc* distinto de 0001".

$$\Pi_{dni}(VENTAS) - \Pi_{dni}(\sigma_{cifc <> 0001}(VENTAS))$$

dni
0002

♦ Problema 2.61

Obtener los nombres de los clientes que no han comprado coches de color 'rojo' en concesionarios de 'Madrid'.

Solución:

El problema se puede replantear como: "obtener el nombre de los clientes cuyo *dni* no aparece en ninguna tupla de la relación VENTAS junto al *color* 'rojo' y a un *cifc* de concesionarios de Madrid".

$$REUNION \leftarrow VENTAS * CONCESIONARIOS$$

$$SELECCION \leftarrow \sigma_{color = 'rojo' \text{ and } ciudad = 'Madrid'}(REUNION)$$

$$PROYECCION \leftarrow \Pi_{dni}(SELECCION)$$

$$DIFERENCIA \leftarrow \Pi_{dni}(CLIENTES) - PROYECCION$$

$$\Pi_{nombre}(DIFERENCIA * CLIENTES)$$

nombre
Juan
Maria
Javier
Ana

♦ Problema 2.62

Obtener el nombre de los clientes que sólo han comprado coches en el concesionario con *cifc* igual a '0001'.

Solución:

Este problema se puede enunciar como: "obtener el nombre de los clientes cuyo *dni* no aparece en ninguna tupla de la relación VENTAS junto a un *cifc* distinto de 0001".

PROY_SELEC ← Π_{DNI} ($\sigma_{CIFC <> 0001}$ (VENTAS))

DIFERENCIA ← Π_{DNI} (VENTAS) - PROY_SELEC

Π_{NOMBRE} (DIFERENCIA * CLIENTES)

nombre
Antonio

♦ **Problema 2.63**

Obtener los valores del atributo *codcoche* de aquellos automóviles que han sido comprados por todos los clientes de 'Madrid'.

Solución:

Para resolver el problema hay que utilizar las relaciones CLIENTES y VENTAS. En primer lugar se obtienen los valores de *dni* de todos los clientes de 'Madrid'. A continuación se calculan los *codcoche* de los coches vendidos a cada cliente. Por último se determinan los *codcoche* correspondientes a los coches que han sido comprados por todos clientes de 'Madrid'.

PROY_SELEC ← Π_{dni} ($\sigma_{ciudad = 'madrid'}$ (CLIENTES))

PROY_REUNION ← $\Pi_{codcoche, dni}$ (VENTAS * CLIENTES)

PROY_REUNION ÷ PROY_SELEC

codcoche

♦ **Problema 2.64**

Obtener el *dni* de los clientes que han adquirido por lo menos los mismos automóviles que el cliente de nombre 'Luis' y apellido 'García'.

Solución:

Considerando las relaciones CLIENTES y VENTAS, se obtienen las tuplas de la relación CLIENTES para 'Luis García' y los *codcoche* de los coches

vendidos a éste. Por último se determinan los valores de *dni* correspondientes a los clientes que han comprado al menos los mismos coches que 'Luis García'.

SELECCION ← $\sigma_{\text{nombre = 'Luis' and apellido = 'Garcia'}}$ (CLIENTES)

PROY_REUNION ← Π_{codcoche} (VENTAS * SELECCION)

$\Pi_{\text{dni, codcoche}}$ (VENTAS) ÷ PROY_REUNION

dni
0001

♦ Problema 2.65

Obtener el valor del atributo *cifc* para los concesionarios que han vendido el mismo coche a todos los clientes.

Solución:

De nuevo, en este problema hay que utilizar las relaciones CLIENTES y VENTAS. En primer lugar se obtienen todos los valores de *dni* de los clientes. A continuación se determinan las tuplas (*cifc*, *codcoche*, *dni*) para todas las ventas. Por último se determinan los valores de *cifc* correspondientes a los concesionarios que han vendido un mismo coche a todos los clientes.

PROYECCION1 ← Π_{dni} (CLIENTES)

PROYECCION2 ← $\Pi_{\text{cifc, codcoche, dni}}$ (VENTAS)

Π_{cifc} (PROYECCION2 ÷ PROYECCION1)

cifc

Capítulo 3
Cálculo Relacional de Tuplas

3.1 Introducción

En el capítulo 2 se presentó el *álgebra relacional* como un procedimiento para manipular los datos dentro del *modelo relacional*. Las consultas, en álgebra relacional, se realizan mediante una serie de operaciones que permiten expresar el procedimiento para construir una determinada relación a partir de las relaciones que componen la base de datos.

El *cálculo relacional* proporciona otro método para manipular los datos que, a diferencia del álgebra relacional, permite definir la relación deseada sin dar un procedimiento específico para obtenerla. El álgebra relacional y el cálculo relacional son totalmente equivalentes en el sentido de que para una expresión del álgebra existe una expresión del cálculo y viceversa.

En el cálculo relacional existen dos enfoques: el *cálculo relacional de tuplas* y el *cálculo relacional de dominios*. En el presente capítulo se proponen y resuelven problemas del cálculo relacional de tuplas y se realiza un recordatorio de los aspectos teóricos más importantes en relación con los problemas.

3.2 Cálculo relacional de tuplas

En el cálculo relacional de tuplas una *consulta* se expresa como el conjunto de todas las tuplas t, tal que el predicado P es verdadero para t:

$$\{t \mid P(t)\}$$

En la expresión anterior, t representa una *variable de tupla* y P una *fórmula* en la que pueden aparecer otras variables de tupla además de t.

Una *variable de tupla* t es una variable que recorre una relación R, es decir, una variable que tiene como únicos valores permitidos las tuplas de la relación R.

Las variables de tuplas pueden ser *libres* o *acotadas*. Una variable de tupla es *libre* cuando no se encuentra cuantificada por un $\exists$ (existe) o por un $\forall$ (para todo). En caso contrario la variable se denomina *acotada*.

Por otra parte, los componentes de una fórmula pueden tener una de las siguientes formas:

- *t pertenece a R*:

  $$t \in R$$

  donde t es una variable de tupla y R una relación. La expresión anterior indica que el valor de la variable de tupla t es una tupla de la relación R.

- *$t[A_1]$ operado con $s[A_2]$* :

  $$t[A_1] \Theta s[A_2]$$

  donde t y s son variables de tupla, A_1 y A_2 son atributos sobre los que están definidas t y s, respectivamente, y Θ es un operador de comparación ($<$, $\leq$, $=$, $>$, $\geq$). Denotando $t[A_1]$ el valor de la variable t en el atributo A_1 y $s[A_2]$ el valor de la variable s en el atributo A_2. Los atributos A_1 y A_2 deben pertenecer a dominios cuyos elementos puedan compararse mediante el operador Θ.

- *$t[A_i]$ operado con c*:

  $$t[A_i] \Theta c$$

  donde t es una variable de tupla, A_i es un atributo sobre el que está definida t, Θ es un operador de comparación y c es una constante en el dominio del atributo A_i.

Para construir una fórmula, a partir de los componentes anteriores, se deben tener en cuenta las siguientes reglas:

- Cada uno de los componentes descritos es una fórmula.
- Si P_1 es una fórmula, entonces también lo son $\neg P_1$ y (P_1).
- Si P_1 y P_2 son fórmulas, entonces también lo son $P_1 \vee P_2$, $P_1 \wedge P_2$, y $P_1 \Rightarrow P_2$.
- Si P(t) es una fórmula que contiene una variable de tupla libre t y R representa una relación, entonces $\exists\, t \in R\ (P(t))$ y $\forall\, t \in R\ (P(t))$ también son fórmulas.

La expresión:

$$\exists\, t \in R\ (P(t))$$

indica que, existe una tupla t en la relación R tal que el predicado P(t) es verdadero.

Por otra parte, la expresión:

$$\forall\, t \in R\ (P(t))$$

denota que, el predicado P(t) es verdadero para todas las tuplas t de la relación R.

Cuando se utilizan las expresiones $P_1 \wedge P_2$, $\forall\, t \in R\ (P(t))$ y $P_1 \Rightarrow P_2$, en una fórmula, éstas se pueden cambiar por sus respectivas expresiones equivalentes:

$$\neg\ (\neg\ P_1 \vee \neg\ P_2) \text{ en lugar de } P_1 \wedge P_2$$

$$\neg\ \exists\, t \in R\ (\neg\ P(t)) \text{ en lugar de } \forall\, t \in R\ (P(t))$$

$$\neg\ P_1 \vee P_2 \text{ en lugar de } P_1 \Rightarrow P_2$$

3.3 Problemas resueltos

En esta sección se proponen y resuelven problemas utilizando el cálculo relacional de tuplas. Mientras no se diga lo contrario, en los problemas se hace referencia a la base de datos AUTOMÓVILES descrita en el apéndice A.

♦ Problema 3.1

Obtener todas las tuplas de la relación CLIENTES.

Solución:

$$\{t \mid t \in \text{CLIENTES}\}$$

Conjunto de todas las tuplas t que verifican la condición de pertenecer a la relación CLIENTES.

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	García	Madrid
0002	Antonio	López	Valencia
0003	Juan	Martín	Madrid
0004	María	García	Madrid
0005	Javier	González	Barcelona
0006	Ana	López	Barcelona

♦ Problema 3.2

Obtener todas las tuplas de la relación DISTRIBUCION cuyo valor en el atributo *cantidad* sea mayor que 10.

Solución:

$$\{t \mid t \in \text{DISTRIBUCION} \wedge t[\text{cantidad}] > 10\}$$

Conjunto de todas las tuplas t que verifican la condición de pertenecer a la relación DISTRIBUCION y que además toman un valor en el atributo *cantidad* mayor que 10.

cifc	*codcoche*	*cantidad*
0005	0016	20

♦ Problema 3.3

Obtener los valores del atributo *cifc* que aparecen junto a un valor del atributo *cantidad* mayor que 10 en las tuplas de la relación DISTRIBUCION.

Solución:

El problema es similar al anterior salvo en que sólo se quieren los valores del atributo *cifc*.

$$\{t \mid \exists\, s \in \text{DISTRIBUCION}\ (t[\text{cifc}] = s[\text{cifc}] \wedge s[\text{cantidad}] > 10)\}$$

Conjunto de todas las tuplas t tal que, para cada una de ellas, existe una tupla s en la relación DISTRIBUCION para la cual los valores de t y s en el atributo *cifc* son iguales y el valor de s en el atributo *cantidad* es mayor que 10.

La variable de tupla t se define sólo en el atributo *cifc* puesto que es el único atributo para el que se especifica una condición para t.

La condición t[cifc] = s[cifc] indica que la relación que recorre la variable de tupla t tiene como único atributo *cifc* y que los valores permitidos para t son los que aparecen en la relación DISTRIBUCION para el atributo *cifc*, es decir:

cifc
0001
0002
0003
0004
0005

Por otra parte, la condición s[cantidad] > 10 restringe la tabla anterior a los valores de *cifc* que aparecen en alguna tupla de la relación DISTRIBUCION junto a un valor del atributo *cantidad* mayor que 10.

Como la única tupla que verifica esta última condición es:

cifc	*codcoche*	*cantidad*
0005	0016	20

el resultado de evaluar la expresión indicada es:

cifc
0005

♦ Problema 3.4

Para el concesionario con *cifc* igual '0005', obtener los valores del atributo *codcoche* que en las tuplas de la relación DISTRIBUCION aparecen junto a un valor del atributo *cantidad* mayor que 10.

Solución:

Este problema difiere del anterior en que en lugar de los valores del atributo *cifc* se quieren los del atributo *codcoche* imponiéndose además una nueva condición sobre *cifc*.

$\{t \mid \exists s \in \text{DISTRIBUCION}\ (t[\text{codcoche}] = s[\text{codcoche}] \wedge s[\text{cantidad}] > 10$

$\wedge\ s[\text{cifc}] = 0005)\}$

Conjunto de todas las tuplas t tal que, para cada una de ellas, existe una tupla s en la relación DISTRIBUCION para la cual los valores de t y s en el atributo *codcoche* son iguales, el valor de s en el atributo *cantidad* es mayor que 10 y el valor de s en el atributo *cifc* es igual a '0005'.

La variable de tupla t se define sólo en el atributo *codcoche* puesto que es el único atributo para el que se especifica una condición para t.

La condición t[codcoche] = s[codcoche] indica que la relación que recorre la variable de tupla t tiene como único atributo *codcoche* y que los valores permitidos para t son los que aparecen en la relación DISTRIBUCION para el atributo *codcoche*, es decir:

codcoche
0001
0005
0006
0008
0009
0010
0011

(Continuación)

codcoche
0012
0013
0014
0015
0016
0017

Por otra parte, las condiciones s[cantidad] > 10 y s[cifc] = 0005 restringen la tabla anterior a los valores de *codcoche* que aparecen en alguna tupla de la relación DISTRIBUCION junto a un valor del atributo *cantidad* mayor que 10 y a un valor del atributo *cifc* igual a '0005'.

Como la única tupla que verifica esta última condición es:

cifc	*codcoche*	*cantidad*
0005	0016	20

el resultado de evaluar la expresión indicada es:

codcoche
0016

♦ Problema 3.5

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo'.

Solución:

En este caso hay que utilizar las relaciones VENTAS y CLIENTES.

$\{t \mid \exists s \in VENTAS\ (t[dni] = s[dni] \wedge s[color] = 'rojo'$

$\wedge \exists u \in CLIENTES\ (u[dni] = s[dni] \wedge t[ciudad] = u[CLIENTES.ciudad]))\}$

Conjunto de todas las tuplas t tal que, para cada una de ellas:

- Existe una tupla s en la relación VENTAS para la cual los valores de t y s en el atributo *dni* son iguales y el valor de s en el atributo *color* es 'rojo'.
- Y existe otra tupla u en la relación CLIENTES para la cual los valores de u y s en el atributo *dni* son iguales y los valores de t y u en el atributo *ciudad* son los mismos.

La variable de tupla s asegura que el cliente ha comprado un coche de color 'rojo' y la variable de tupla u, que tiene la restricción de corresponder al mismo cliente que s, asegura que la ciudad asignada a la variable de tupla t es la del cliente.

Como en la base de datos AUTOMÓVILES, descrita en el apéndice A, existen los atributos *nombre* y *ciudad* en más de una relación refiriéndose a entidades distintas, cuando se haga referencia a dichos atributos se los antepondrá el nombre de la relación. Así, en la solución de este problema se ha expresado u[CLIENTES.ciudad] en lugar de u[ciudad].

La variable de tupla t se define en los atributos *dni* y *ciudad* por ser los únicos atributos para los que se especifican condiciones para t.

Las condiciones t[dni] = s[dni] y t[ciudad] = u[CLIENTES.ciudad] indican que la relación que recorre la variable de tupla t tiene como atributos *dni* y *ciudad*, siendo los valores permitidos de *dni* los que aparecen en la relación VENTAS (0001, 0002, 0003, 0004, 0005) y los valores permitidos de *ciudad* los que aparecen en la relación CLIENTES (Madrid, Valencia, Barcelona).

Como las tuplas de la relación VENTAS tienen que verificar la condición s[color] = 'rojo', los valores de *dni* se reducen a (0001, 0002, 0004).

Por otra parte, como las tuplas de la relación CLIENTES tienen que verificar la condición u[dni] = s[dni], los valores de *ciudad* se restringen a (Madrid, Valencia).

Por último, la condición t[ciudad] = u[CLIENTES.ciudad], hace que a cada valor de *dni* se le asigne la correspondiente ciudad del cliente.

dni	*ciudad*
0001	Madrid
0002	Valencia
0004	Madrid

♦ Problema 3.6

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo' o de color 'blanco' o de ambos colores.

Solución:

El problema es similar al anterior salvo en que el color del coche puede ser 'rojo' o 'blanco'. Para imponer esta condición, en la expresión del problema

anterior, se ha sustituido s[color] = 'rojo' por (s[color] = 'rojo' ∨ s[color] = 'blanco').

{t | ∃ s ∈ VENTAS (t[dni] = s[dni] ∧ (s[color] = 'rojo' ∨ s[color] = 'blanco')

∧ ∃ u ∈ CLIENTES (u[dni] = s[dni] ∧ t[ciudad] = u[CLIENTES.ciudad]))}

dni	*Ciudad*
0001	Madrid
0002	Valencia
0003	Madrid
0004	Madrid

♦ Problema 3.7

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado al menos un coche de color 'rojo' y otro coche de color 'blanco'.

Solución:

Este problema difiere del anterior en que los clientes deben haber adquirido al menos un coche 'rojo' y otro 'blanco'.

En principio se puede pensar que la expresión que resuelve el problema es la misma que la indicada en el problema anterior sustituyendo (s[color] = 'rojo' ∨ s[color] = 'blanco') por (s[color] = 'rojo' ∧ s[color] = 'blanco'), es decir, cambiando ∨ ('o') por ∧ ('y').

Obsérvese que (s[color] = 'rojo' ∨ s[color] = 'blanco') es verdad cuando la variable de tupla s toma en el atributo *color* el valor 'rojo' o 'blanco', mientras que (s[color] = 'rojo' ∧ s[color] = 'blanco') es verdad si la variable de tupla s toma en el atributo *color* los valores 'rojo' y 'blanco' simultáneamente.

Como quiera que una variable de tupla no puede tomar simultáneamente dos valores en un mismo atributo, es necesario considerar otra variable de tupla adicional para imponer la nueva condición como se indica en la siguiente expresión.

{t | ∃ s ∈ VENTAS (t[dni] = s[dni] ∧ s[color] = 'rojo'

∧ ∃ v ∈ VENTAS (v[dni] = s[dni] ∧ v[color] = 'blanco'

∧ ∃ u ∈ CLIENTES (u[dni] = v[dni] ∧ t[ciudad] = u[CLIENTES.ciudad])))}

Conjunto de todas las tuplas t tal que, para cada una de ellas:

- Existe una tupla s en la relación VENTAS para la cual los valores de t y s en el atributo *dni* son iguales y el valor de s en el atributo *color* es 'rojo'.
- Y existe otra tupla v en la relación VENTAS para la cual los valores de v y s en el atributo *dni* son iguales y el valor de v en el atributo *color* es 'blanco'. Con ello se asegura que el cliente ha comprado un coche 'rojo' y otro 'blanco'.
- Y existe otra tupla u en la relación CLIENTES para la cual los valores de u y v en el atributo *dni* son iguales y los valores de t y u en el atributo *ciudad* son los mismos. Con ello se asegura que la *ciudad* asignada a t es la del cliente que verifica la condición impuesta en el enunciado.

La variable de tupla t se define en los atributos *dni* y *ciudad* por ser los únicos atributos para los que se especifican condiciones para t.

dni	*ciudad*
0001	Madrid

♦ Problema 3.8

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo' y alguno que no sea de color 'blanco'.

Solución:

El problema es similar al anterior imponiendo la condición de que los clientes deben haber adquirido al menos un coche 'rojo' y otro que no sea 'blanco'. Para ello, en la expresión del problema anterior, se sustituye v[color] = 'blanco' por su negación ¬ (v[color] = 'blanco').

$$\{t \mid \exists s \in VENTAS\ (t[dni] = s[dni] \wedge s[color] = 'rojo'$$

$$\wedge \exists v \in VENTAS\ (v[dni] = s[dni] \wedge \neg\ (v[color] = 'blanco')$$

$$\wedge \exists u \in CLIENTES\ (u[dni] = v[dni] \wedge t[ciudad] = u[CLIENTES.ciudad]))\}$$

Obsérvese que para un cliente que hubiera adquirido tres coches de colores 'rojo', 'verde' y 'blanco' en la relación resultante aparecerían sus valores de *dni* y *ciudad*.

dni	*ciudad*

♦ Problema 3.9

Obtener los valores del atributo *dni* para los clientes que han comprado coches de color 'rojo' pero no han comprado coches de color 'blanco'.

Solución:

El problema es similar a los anteriores imponiendo la condición de que en la relación resultante deben aparecer los valores del atributo *dni* de aquellos clientes que habiendo comprado al menos un coche de color 'rojo' no han comprado ninguno de color 'blanco'. Para ello, en la expresión del problema 3.7 se sustituye $\exists$ v $\in$ VENTAS (...) por $\neg \exists$ v $\in$ VENTAS (...) y se elimina $\exists$ u $\in$ CLIENTES (...) por no ser necesario utilizar la relación CLIENTES.

$\{t \mid \exists s \in \text{VENTAS}\ (t[\text{dni}] = s[\text{dni}] \wedge s[\text{color}] = \text{'rojo'}$

$\wedge \neg \exists v \in \text{VENTAS}\ (v[\text{dni}] = s[\text{dni}] \wedge v[\text{color}] = \text{'blanco'}))\}$

Conjunto de todas las tuplas t tal que, para cada una de ellas:

- Existe una tupla s en la relación VENTAS para la cual los valores de t y s en el atributo *dni* son iguales y el valor de s en el atributo *color* es 'rojo'.
- Y no existe otra tupla v en la relación VENTAS para la cual los valores de v y s en el atributo *dni* son iguales y el valor de v en el atributo *color* es 'blanco'. Con ello se asegura que el cliente que ha comprado un coche 'rojo' no ha comprado otro 'blanco'.

Obsérvese que para un cliente que hubiera adquirido tres coches de colores 'rojo', 'verde' y 'blanco' en la relación resultante no aparecería su *dni*.

dni
0002
0004

♦ Problema 3.10

Obtener los valores del atributo *dni* para los clientes que no han comprado coches de color 'rojo' o han comprado al menos un coche de color 'blanco'.

Solución:

$\{t \mid \neg \exists s \in VENTAS\ (t[dni] = s[dni] \wedge s[color] = 'rojo')$

$\vee \exists v \in VENTAS\ (t[dni] = v[dni] \wedge v[color] = 'blanco')\}$

Conjunto de todas las tuplas t tal que, para cada una de ellas:

- No existe ninguna tupla s en la relación VENTAS para la cual los valores de t y s en el atributo *dni* son iguales y el valor de s en el atributo *color* es 'rojo'.
- O existe al menos una tupla v en la relación VENTAS para la cual los valores de t y v en el atributo *dni* son iguales y el valor de v en el atributo *color* es 'blanco'.

Obsérvese que la condición "no han comprado coches de color 'rojo' o han comprado al menos un coche de color 'blanco' ($\neg P_1 \vee P_2$)" es equivalente a la condición "si han comprado coches de color 'rojo' entonces han comprado al menos un coche de color 'blanco' ($P_1 \Rightarrow P_2$)". Por lo que la expresión anterior se puede poner en la forma:

$\{t \mid \exists s \in VENTAS\ (t[dni] = s[dni] \wedge s[color] = 'rojo')$

$\Rightarrow \exists v \in VENTAS\ (t[dni] = v[dni] \wedge v[color] = 'blanco')\}$

dni
0001
0003
0005

♦ Problema 3.11

Obtener el nombre de los clientes que han comprado coches en todos los concesionarios.

Solución:

En este caso es necesario utilizar las relaciones CONCESIONARIOS, VENTAS y CLIENTES.

$\{t \mid \forall s \in CONCESIONARIOS\ (\exists v \in VENTAS\ (v[cifc] = s[cifc]$

$\wedge \exists u \in CLIENTES\ (u[dni] = v[dni] \wedge t[nombre] = u[CLIENTES.nombre]))\}$

Conjunto de todas las tuplas t tal que, para todas las tuplas s pertenecientes a la relación CONCESIONARIOS:

- Existe una tupla v perteneciente a la relación VENTAS, con los valores de v y s en el atributo *cifc* iguales
- Y existe una tupla u perteneciente a la relación CLIENTES, con los valores de u y v en el atributo *dni* iguales, siendo los valores de t los de u en el atributo *nombre*.

nombre

♦ Problema 3.12

Sean COCHES1 y COCHES2 dos relaciones en las que aparecen las tuplas de la relación COCHES correspondientes al modelo 'gti' y las tuplas de la relación COCHES que tienen por nombre 'ibiza', respectivamente. Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional COCHES1 $\cup$ COCHES2.

COCHES1

codcoche	*nombre*	*modelo*
0002	ibiza	gti
0005	cordoba	gti
0007	megane	gti
0016	astra	gti

COCHES2

codcoche	*nombre*	*modelo*
0001	ibiza	glx
0002	ibiza	gti
0003	ibiza	gtd

Solución:

$$\{t \mid t \in COCHES1 \vee t \in COCHES2\}$$

Conjunto de todas las tuplas t tal que cada una de ellas pertenece a la relación COCHES1 o pertenece a la relación COCHES2.

codcoche	*nombre*	*modelo*
0002	ibiza	gti
0005	cordoba	gti
0007	megane	gti
0016	astra	gti
0001	ibiza	glx
0003	ibiza	gtd

♦ Problema 3.13

Sean COCHES1 y COCHES2 las relaciones indicadas en el problema 3.12. Dar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional COCHES1 ∩ COCHES2.

Solución:

$$\{t \mid t \in \text{COCHES1} \wedge t \in \text{COCHES2}\}$$

Conjunto de todas las tuplas t tal que cada una de ellas pertenece a la relación COCHES1 y a la relación COCHES2.

codcoche	*nombre*	*modelo*
0002	ibiza	gti

♦ Problema 3.14

Sean COCHES1 y COCHES2 las relaciones indicadas en el problema 3.12. Dar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional COCHES1 - COCHES2.

Solución:

$$\{t \mid t \in \text{COCHES1} \wedge \neg (t \in \text{COCHES2})\}$$

Conjunto de todas las tuplas t tal que cada una de ellas pertenece a la relación COCHES1 pero no pertenece a la relación COCHES2.

codcoche	*nombre*	*modelo*
0005	cordoba	gti
0007	megane	gti
0016	astra	gti

♦ Problema 3.15

Sean NMARCAS y NCONCESION dos relaciones en las que aparecen los nombres de las marcas de coches y los nombres de los concesionarios, respectivamente. Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional producto cartesiano NMARCAS × NCONCESION.

NMARCAS

mnombre
seat
renault
citroen
audi
opel
bmw

NCONCESION

cnombre
acar
bcar
ccar
dcar
ecar

Solución:

$\{t \mid \exists s \in NMARCAS\ (t[mnombre] = s[mnombre])$

$\wedge \exists v \in NCONCESION\ (t[cnombre] = v[cnombre])\}$

Conjunto de todas las tuplas t tal que, para cada una de ellas:

- Existe una tupla s en la relación NMARCAS para la cual los valores de t y s en el atributo *mnombre* son iguales.
- Y existe otra tupla v en la relación NCONCESION para la cual los valores de t y v en el atributo *cnombre* son iguales.

mnombre	*cnombre*
seat	acar
seat	bcar
seat	ccar
seat	dcar
seat	ecar
renault	acar
renault	bcar
renault	ccar
renault	dcar
renault	ecar

(continuación)

mnombre	*cnombre*
citroen	acar
citroen	bcar
citroen	ccar
citroen	dcar
citroen	ecar
audi	acar
audi	bcar
audi	ccar
audi	dcar
audi	ecar

(continuación)

mnombre	*cnombre*
opel	acar
opel	bcar
opel	ccar
opel	dcar
opel	ecar
bmw	acar
bmw	bcar
bmw	ccar
bmw	dcar
bmw	ecar

Obsérvese como la relación resultante está formada por el conjunto de todas las tuplas t tales que t es la combinación de una tupla s perteneciente a la relación NMARCAS y una tupla v perteneciente a la relación NCONCESION.

♦ Problema 3.16

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional $\sigma_{\text{ciudad = 'Barcelona'}}$ (MARCAS).

Solución:

$$\{t \mid t \in \text{MARCAS} \wedge t[\text{ciudad}] = \text{'Barcelona'}\}$$

Conjunto todas las tuplas t que verifican la condición de pertenecer a la relación MARCAS y que además toman un valor en el atributo *ciudad* igual a 'Barcelona'.

cifm	*nombre*	*ciudad*
0002	renault	Barcelona
0006	bmw	Barcelona

♦ Problema 3.17

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional $\sigma_{\text{apellido = 'García' AND ciudad = 'Madrid'}}$ (CLIENTES).

Solución:

$$\{t \mid t \in \text{CLIENTES} \wedge t[\text{apellido}] = \text{'García'} \wedge t[\text{ciudad}] = \text{'Madrid'}\}$$

Conjunto todas las tuplas t que verifican la condición de pertenecer a la relación CLIENTES y que además toman un valor en el atributo *apellido* igual a 'García' y toman un valor en el atributo *ciudad* igual a 'Barcelona'.

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	García	Madrid
0004	María	García	Madrid

♦ Problema 3.18

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional Π_{apellido} (CLIENTES).

Solución:

$$\{t \mid \exists\, s \in \text{CLIENTES}\ (t[\text{apellido}] = s[\text{apellido}])\}$$

Conjunto de todas las tuplas t tal que, para cada una de ellas, existe una tupla s en la relación CLIENTES para la cual los valores de t y s en el atributo *apellido* son iguales.

apellido
García
López
Martín
González

♦ Problema 3.19

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional $\Pi_{\text{apellido, ciudad}}$ (CLIENTES).

Solución:

$$\{t \mid \exists\, s \in \text{CLIENTES}\ (t[\text{apellido}] = s[\text{apellido}]$$

$$\wedge\ t[\text{ciudad}] = s[\text{CLIENTES.ciudad}])\}$$

Conjunto de todas las tuplas t tal que, para cada una de ellas, existe una tupla s en la relación CLIENTES para la cual los valores de t y s en los atributos *apellido* y *ciudad* son iguales.

apellido	*ciudad*
García	Madrid
López	Valencia
Martín	Madrid
González	Barcelona
López	Barcelona

♦ Problema 3.20

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional $\Pi_{\text{apellido, ciudad}}$ ($\sigma_{\text{ciudad = 'Madrid'}}$ (CLIENTES)).

Solución:

$\{t \mid \exists s \in CLIENTES\ (s[CLIENTES.ciudad] = 'Madrid'$

$\wedge\ t[apellido] = s[apellido] \wedge t[ciudad] = s[CLIENTES.ciudad])\}$

Conjunto de todas las tuplas t tal que, para cada una de ellas, existe una tupla s en la relación CLIENTES para la cual el valor en el atributo *ciudad* es 'Madrid' y los valores de t y s en los atributos *apellido* y *ciudad* son iguales.

apellido	*ciudad*
García	Madrid
Martín	Madrid

♦ Problema 3.21

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional (MARCAS $\rho_{mnombre}$ (NOMBRE)) * CLIENTES (reunión natural de las relaciones MARCAS y CLIENTES según el atributo *ciudad*).

Solución:

$\{t \mid \exists s \in MARCAS\ (t[cifm] = s[cifm] \wedge t[mnombre] = s[MARCAS.nombre]$

$\wedge \exists v \in CLIENTES\ (v[CLIENTES.ciudad] = s[MARCAS.ciudad]$

$\wedge\ t[dni] = v[dni] \wedge t[nombre] = v[CLIENTES.nombre] \wedge t[apellido] = v[apellido]$

$\wedge\ t[ciudad] = v[CLIENTES.ciudad]))\}$

cifm	*mnombre*	*dni*	*nombre*	*apellido*	*ciudad*
0001	seat	0001	Luis	García	Madrid
0001	seat	0003	Juan	Martín	Madrid
0001	seat	0004	María	García	Madrid
0002	renault	0005	Javier	González	Barcelona
0002	renault	0006	Ana	López	Barcelona
0003	citroen	0002	Antonio	López	Valencia
0004	audi	0001	Luis	García	Madrid
0004	audi	0003	Juan	Martín	Madrid
0004	audi	0004	María	García	Madrid
0006	bmw	0005	Javier	González	Barcelona
0006	bmw	0006	Ana	López	Barcelona

Obsérvese que la condición v[CLIENTES.ciudad] = s[MARCAS.ciudad] es la que obliga a que la reunión se realice por el atributo común *ciudad*.

♦ Problema 3.22

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional (MARCAS $\rho_{\text{mnombre, mciudad}}$ (nombre, ciudad)) $|\times|_{\text{mciudad} > \text{ciudad}}$ CLIENTES (reunión 'mayor que' de la relación MARCAS según el atributo *ciudad* con la relación CLIENTES según el atributo *ciudad*).

Solución:

Este problema se resuelve de forma similar al anterior cambiando la condición v[CLIENTES.ciudad] = s[MARCAS.ciudad] por s[MARCAS.ciudad] > v[CLIENTES.ciudad] y haciendo que aparezcan en la relación resultante las ciudades de las marcas y de los clientes.

$\{t \mid \exists\, s \in \text{MARCAS}\ (t[\text{cifm}] = s[\text{cifm}] \wedge t[\text{mnombre}] = s[\text{MARCAS.nombre}]$

$\wedge\ t[\text{mciudad}] = s[\text{MARCAS.ciudad}]$

$\wedge\ \exists\, v \in \text{CLIENTES}\ (s[\text{MARCAS.ciudad}] > v[\text{CLIENTES.ciudad}]$

$\wedge\ t[\text{dni}] = v[\text{dni}] \wedge t[\text{nombre}] = v[\text{CLIENTES.nombre}] \wedge t[\text{apellido}] = v[\text{apellido}]$

$\wedge\ t[\text{ciudad}] = v[\text{CLIENTES.ciudad}]))\}$

cifm	*mnombre*	*mciudad*	*dni*	*nombre*	*apellido*	*ciudad*
0001	seat	Madrid	0005	Javier	González	Barcelona
0001	seat	Madrid	0006	Ana	López	Barcelona
0003	citroen	Valencia	0001	Luis	García	Madrid
0003	citroen	Valencia	0003	Juan	Martín	Madrid
0003	citroen	Valencia	0004	María	García	Madrid
0003	citroen	Valencia	0005	Javier	González	Barcelona
0003	citroen	Valencia	0006	Ana	López	Barcelona
0004	audi	Madrid	0005	Javier	González	Barcelona
0004	audi	Madrid	0006	Ana	López	Barcelona
0005	opel	Bilbao	0005	Javier	González	Barcelona
0005	opel	Bilbao	0006	Ana	López	Barcelona

♦ Problema 3.23

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional DIVIDENDO ÷ DIVISOR.

DIVIDENDO

atributo1	*atributo2*
dato1	dato6
dato1	dato8
dato1	dato9
dato2	dato6
dato3	dato5
dato3	dato6

(continuación)

atributo1	*atributo2*
dato4	dato10
dato4	dato9
dato4	dato8
dato4	dato7
dato4	dato6
dato4	dato5

DIVISOR

atributo2
dato6
dato8

Solución:

$\{t \mid \forall s \in DIVISOR ($

$\exists v \in DIVIDENDO (t[atributo1] = v[atributo1]$

$\wedge v[DIVIDENDO.atributo2] = s[DIVISOR.atributo2]]))\}$

Conjunto de todas las tuplas t tal que, para todas las tuplas s pertenecientes a la relación DIVISOR existe una tupla v perteneciente a la relación DIVIDENDO con los valores de t y v iguales en el *atributo1* y con los valores de v y s iguales en el *atributo2*.

atributo1
dato1
dato4

♦ Problema 3.24

Obtener los nombres de las marcas que tienen modelos 'gtd'.

Solución:

En el álgebra relacional el problema se puede resolver con las siguientes operaciones:

- Selección de las tuplas de la relación COCHES en las que el atributo *modelo* toma el valor 'gtd'.

- Reunión natural de la relación anterior con la relación MARCO según el atributo común *codcoche*.
- Reunión natural de la relación anterior con la relación COCHES según el atributo común *cifm*.
- Proyección por el atributo *nombre* de la relación MARCAS.

Esto equivale en el cálculo relacional de tuplas a la siguiente expresión.

$\{t \mid \exists s \in COCHES\ (s[modelo] = 'gtd'$

$\wedge \exists v \in MARCO\ (v[codcoche] = s[codcoche]$

$\wedge \exists u \in MARCAS\ (u[cifm] = v[cifm] \wedge t[nombre] = u[MARCAS.nombre]))))\}$

Conjunto de todas las tuplas t tal que, para cada una de ellas:

- Existe un tupla s en la relación COCHES para la cual el valor de s en el atributo *modelo* es 'gtd'.
- Y existe otra tupla v en la relación MARCO para la cual los valores de v y s en el atributo *codcoche* son iguales.
- Y existe otra tupla u en la relación MARCAS para la cual los valores de u y v en el atributo *cifm* son iguales y los valores de t y u en el atributo *nombre* son iguales.

nombre
seat
renault
citroen

♦ Problema 3.25

Obtener los nombres de las marcas de las que se han vendido coches de color 'rojo'.

Solución:

Este problema se resuelve de forma similar al anterior utilizando las relaciones VENTAS, MARCO y MARCAS.

{t | ∃ s ∈ VENTAS (s[color] = 'rojo'

∧ ∃ v ∈ MARCO (v[codcoche] = s[codcoche]

∧ ∃ u ∈ MARCAS (u[cifm] = v[cifm] ∧ t[nombre] = u[MARCAS.nombre])))}

nombre
seat
renault
citroen

♦ Problema 3.26

Obtener los nombres de los coches que tengan al menos todos los modelos que tiene el coche de nombre 'cordoba'.

Solución:

En este caso sólo es necesario utilizar la relación COCHES.

En álgebra relacional el problema se resuelve mediante la división de las relaciones DIVIDENDO entre DIVISOR, siendo DIVIDENDO la relación resultante de proyectar la relación COCHES, según los atributos *nombre* y *modelo*, y DIVISOR la relación que resulta al proyectar, según el atributo *modelo*, las tuplas de la relación COCHES para las que el atributo *nombre* es igual a 'cordoba'.

Esto equivale en cálculo relacional de tuplas a la siguiente expresión:

{t | ∀ s ∈ COCHES ((s[COCHES.nombre] = 'cordoba')

⇒ ∃ v ∈ COCHES (v[modelo] = s[modelo]

∧ t[nombre] = v[COCHES.nombre]))}

Conjunto de todas las tuplas t tal que, para todas las tuplas s pertenecientes a la relación COCHES, si el valor de s en el atributo *nombre* es 'cordoba', entonces existe una tupla v en la relación COCHES para la que el atributo *modelo* toma el mismo valor que aparece en s y t toma el valor del atributo *nombre* de v.

Por otra parte, como $P_1 \Rightarrow P_2$ equivale a $\neg P_1 \vee P_2$, la expresión anterior se puede poner en la forma:

$\{t \mid \forall s \in COCHES\ (\neg\ (s[COCHES.nombre] = 'cordoba')$

$\vee \exists v \in COCHES\ (v[modelo] = s[modelo]$

$\wedge\ t[nombre] = v[COCHES.nombre]))\}$

Conjunto de todas las tuplas t tal que, para todas las tuplas s de la relación COCHES, o el valor de s en el atributo *nombre* es distinto de 'cordoba' o existe una tupla v en la relación COCHES para la que el atributo *modelo* toma el mismo valor que aparece en s y t toma el valor del atributo *nombre* de v.

nombre
ibiza
cordoba
megane
astra

♦ Problema 3.27

Obtener el nombre de los coches que no tengan modelo 'gtd'.

Solución:

El problema se puede replantear como, "obtener el nombre de los coches para los que no existe ninguna tupla en la relación COCHES en la que el atributo *modelo* toma el valor gtd".

$\{t \mid \neg \exists s \in COCHES\ (t[nombre] = s[COCHES.nombre] \wedge s[modelo] = 'gtd')\}$

nombre
cordoba
megane
zx
a4
astra
corsa
300
500
700

♦ Problema 3.28

Obtener todas las tuplas de la relación CONCESIONARIOS.

Solución:

$$\{t \mid t \in \text{CONCESIONARIOS}\}$$

cifc	*nombre*	*ciudad*
0001	acar	Madrid
0002	bcar	Madrid
0003	ccar	Barcelona
0004	dcar	Valencia
0005	ecar	Bilbao

♦ Problema 3.29

Obtener todas las tuplas de todos los clientes de 'Madrid'.

Solución:

$$\{t \mid t \in \text{CLIENTES} \wedge t[\text{ciudad}] = \text{'Madrid'}\}$$

En álgebra relacional esta expresión equivale a seleccionar las tuplas de la relación CLIENTES cuyo valor en el atributo *ciudad* es 'Madrid'.

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	García	Madrid
0003	Juan	Martín	Madrid
0004	María	García	Madrid

♦ Problema 3.30

Obtener todas las parejas de atributos *cifm* de la relación MARCAS y *dni* de la relación CLIENTES que sean de la misma ciudad.

Solución:

$$\{t \mid \exists\, s \in \text{MARCAS}\ (t[\text{cifm}] = s[\text{cifm}] \wedge \exists\, v \in \text{CLIENTES}\ (t[\text{dni}] = v[\text{dni}]$$

$$\wedge\ v[\text{CLIENTES.ciudad}] = s[\text{MARCAS.ciudad}]))\}$$

En álgebra relacional esta expresión equivale a la reunión natural de las relaciones MARCAS y CLIENTES, según el atributo común *ciudad*, proyectada según los atributos *cifm* y *dni*.

cifm	*dni*
0001	0001
0001	0003
0001	0004
0002	0005
0002	0006
0003	0002
0004	0001
0004	0003
0004	0004
0006	0005
0006	0006

♦ Problema 3.31

Obtener todas las parejas de valores de los atributos *cifm* de la relación MARCAS y *dni* de la relación CLIENTES que no sean de la misma ciudad.

Solución:

$\{t \mid \exists\, s \in MARCAS\ (t[cifm] = s[cifm] \wedge \exists\, v \in CLIENTES\ (t[dni] = v[dni]$

$\wedge \neg\ (v[CLIENTES.ciudad] = s[MARCAS.ciudad]))\}$

En álgebra relacional esta expresión equivale a la reunión *'distinto que'* de las relaciones MARCAS y CLIENTES, para el atributo común *ciudad*, proyectada según los atributos *cifm* y *dni*.

cifm	*dni*
0001	0002
0001	0005
0001	0006
0002	0001
0002	0002
0002	0003
0002	0004
0003	0001
0003	0003
0003	0004
0003	0005
0003	0006
0004	0002

(continuación)

cifm	*dni*
0004	0005
0004	0006
0005	0001
0005	0002
0005	0003
0005	0004
0005	0005
0005	0006
0006	0001
0006	0002
0006	0003
0006	0004

♦ Problema 3.32

Obtener los valores del atributo *codcoche* para los coches que se encuentran en algún concesionario de 'Barcelona'.

Solución:

$\{t \mid \exists s \in \text{CONCESIONARIOS}\ (s[\text{CONCESIONARIOS.ciudad}] = \text{'Barcelona'}$

$\wedge \exists v \in \text{DISTRIBUCION}\ (v[\text{cifc}] = s[\text{cifc}] \wedge t[\text{codcoche}] = v[\text{codcoche}]))\}$

codcoche
0010
0011
0012

♦ Problema 3.33

Obtener el valor del atributo *codcoche* de aquellos coches vendidos a clientes de 'Madrid'.

Solución:

$\{t \mid \exists s \in \text{CLIENTES}\ (s[\text{CLIENTES.ciudad}] = \text{'Madrid'}$

$\wedge \exists v \in \text{VENTAS}\ (v[\text{dni}] = s[\text{dni}] \wedge t[\text{codcoche}] = v[\text{codcoche}]))\}$

codcoche
0001
0006
0011
0008

♦ Problema 3.34

Obtener los valores del atributo *codcoche* para los coches que han sido adquiridos por un cliente de 'Madrid' en un concesionario de 'Madrid'.

Solución:

{t | ∃ s ∈ CLIENTES (s[CLIENTES.ciudad] = 'Madrid'

∧ ∃ u ∈ VENTAS (u[dni] = s[dni] ∧ t[codcoche] = u[codcoche]

∧ ∃ v ∈ CONCESIONARIOS (v[cifc] = u[cifc]

∧ v[CONCESIONARIOS.ciudad] = 'Madrid'))))}

codcoche
0001
0008
0006

♦ Problema 3.35

Obtener los valores del atributo *codcoche* para los coches comprados en un concesionario de la misma ciudad que la del cliente que lo compra.

Solución:

El problema se resuelve de forma similar al anterior imponiendo que las ciudades del concesionario y del cliente sean las mismas.

{t | ∃ s ∈ VENTAS (t[codcoche] = s[codcoche]

∧ ∃ v ∈ CLIENTES (v[dni] = s[dni]

∧ ∃ u ∈ CONCESIONARIOS (u[cifc] = s[cifc])

∧ u[CONCESIONARIOS.ciudad] = v[CLIENTES.ciudad])))}

codcoche
0001
0008
0006

♦ Problema 3.36

Obtener los valores del atributo *codcoche* para los coches comprados en un concesionario de distinta ciudad que la del cliente que lo compra.

Solución:

La forma de resolver este problema es similar a la aplicada en el problema anterior, imponiendo que las ciudades del concesionario y del cliente sean distintas.

{t | ∃ s ∈ VENTAS (t[codcoche] = s[codcoche]

∧ ∃ v ∈ CLIENTES (v[dni] = s[dni]

∧ ∃ u ∈ CONCESIONARIOS (u[cifc] = s[cifc])

∧ ¬ (u[CONCESIONARIOS.ciudad] = v[CLIENTES.ciudad])))}

codcoche
0005
0011
0014

♦ **Problema 3.37**

Obtener todas las parejas de nombre de marcas que sean de la misma ciudad.

Solución:

{t | ∃ s ∈ MARCAS (t[nombre] = s[MARCAS.nombre]

∧ ∃u ∈ MARCAS (t[mnombre] = u[MARCAS.nombre]

∧ u[MARCAS.ciudad] = s[MARCAS.ciudad]

∧ u[MARCAS.nombre] < s[MARCAS.nombre]))}

La condición u[MARCAS.nombre] < s[MARCAS.nombre] permite eliminar las parejas (nombre1, nombre1) de nombres iguales y garantizar la aparición de una sola de las parejas (nombre1, nombre2) y (nombre2, nombre1).

nombre	*mnombre*
audi	seat
bmw	renault

♦ Problema 3.38

Obtener las parejas de modelos de coches cuyo nombre es el mismo y cuya marca es de 'Bilbao'.

Solución:

Este problema se resuelve de forma similar al anterior utilizando las relaciones MARCAS, MARCO y COCHES.

{t | ∃ s ∈ MARCAS (s[MARCAS.ciudad] = 'Bilbao'

∧ ∃ u ∈ MARCO (u[cifm] = s[cifm]

∧ ∃ v ∈ COCHES (t[modelo] = v[modelo] ∧ v[codcoche] = u[codcohe]

∧ ∃ w ∈ COCHES (t[nmodelo] = w[modelo]

∧ w[COCHES.nombre] = v[COCHES.nombre] ∧ w[modelo] < v[modelo])))))}

modelo	*nmodelo*
caravan	gti

♦ Problema 3.39

Obtener el *dni* de los clientes que han comprado algún coche en un concesionario de 'Madrid'.

Solución:

{t | ∃ s ∈ CONCESIONARIOS (s[CONCESIONARIOS.ciudad] = 'Madrid'

∧ ∃ v ∈ VENTAS (v[cifc] = s[cifc] ∧ t[dni] = v[dni])}

dni
0001
0002
0003

♦ Problema 3.40

Obtener el color de los coches vendidos por el concesionario 'acar'.

Solución:

$\{t \mid \exists s \in CONCESIONARIOS\ (s[CONCESIONARIOS.nombre] = 'acar'$

$\wedge \exists v \in VENTAS\ (v[cifc] = s[cifc] \wedge t[color] = v[color]))\}$

color
blanco
rojo

♦ Problema 3.41

Obtener el nombre y el modelo de los coches vendidos por algún concesionario de 'Barcelona'.

Solución:

$\{t \mid \exists s \in CONCESIONARIO\ (s[CONCESIONARIO.ciudad] = 'Barcelona'$

$\wedge \exists u \in VENTAS\ (u[cifc] = s[cifc]$

$\wedge \exists v \in COCHES\ (v[codcoche] = u[codcoche]$

$\wedge t[nombre] = v[COCHES.nombre] \wedge t[modelo] = v[modelo])))\}$

nombre	*modelo*
zx	td

♦ Problema 3.42

Obtener todos los nombres de los clientes que hayan adquirido algún coche del concesionario 'dcar'.

Solución:

$\{t \mid \exists s \in CONCESIONARIOS\ (s[CONCESIONARIOS.nombre] = 'dcar'$

$\wedge \exists v \in VENTAS\ (v[cifc] = s[cifc]$

$\wedge \exists u \in CLIENTES\ (u[dni] = u[dni] \wedge t[nombre] = u[CLIENTES.nombre])))\}$

nombre
Javier

♦ Problema 3.43

Obtener el nombre y el apellido de los clientes que han adquirido un coche modelo 'gti' de color 'blanco'.

Solución:

{t | ∃ s ∈ CLIENTES (t[nombre] = s[CLIENTES.nombre]

∧ t[apellido] = s[apellido]

∧ ∃ u ∈ VENTAS (u[dni] = s[dni] ∧ u[color] = 'blanco'

∧ ∃ v ∈ COCHES (v[codcoche] = u[codcoche] ∧ v[modelo] = 'gti'))))}

nombre	*apellido*

♦ Problema 3.44

Obtener el nombre y el apellido de los clientes que han adquirido un automóvil en un concesionario que dispone actualmente de coches del modelo 'gti'.

Solución:

{t | ∃ s ∈ COCHES (s[modelo] = 'gti'

∧ ∃ u ∈ DISTRIBUCION (u[codcoche] = s[codcoche]

∧ ∃ v ∈ VENTAS (v[cifc] = u[cifc]

∧ ∃ w ∈ CLIENTES (w[dni] = v[dni]

∧ t[nombre] = w[CLIENTES.nombre] ∧ t[apellido] = w[apellido])))))}

nombre	*apellido*
Luis	García
Antonio	López

♦ Problema 3.45

Obtener el nombre y el apellido de los clientes que han adquirido un automóvil en un concesionario de 'Madrid' que dispone de coches del modelo 'gti'.

Solución:

$\{t \mid \exists s \in CLIENTES\ (t[nombre] = s[CLIENTES.nombre]$

$\wedge\ t[apellido] = s[apellido]$

$\wedge \exists u \in VENTAS\ (u[dni] = s[dni]$

$\wedge \exists v \in CONCESIONARIOS\ (v[cifc] = u[cifc] \wedge v[ciudad] = 'Madrid'$

$\wedge \exists w \in DISTRIBUCION\ (w[cifc] = v[cifc]$

$\wedge \exists z \in COCHES\ (z[codcoche] = w[codcoche] \wedge z[modelo] = 'gti'))))\}$

nombre	*apellido*
Luis	García

♦ Problema 3.46

Obtener el nombre y el apellido de los clientes cuyo *dni* es menor que el correspondiente al cliente de nombre 'Juan' y apellido 'Martín'.

Solución:

$\{t \mid \exists s \in CLIENTES\ (s[CLIENTES.nombre] = 'Juan'$

$\wedge\ s[apellido] = 'Martín'$

$\wedge \exists u \in CLIENTES\ (u[dni] < s[dni] \wedge t[nombre] = u[CLIENTES.nombre]$

$\wedge\ t[apellido] = u[apellido]))\}$

nombre	*apellido*
Luis	García
Antonio	López

♦ Problema 3.47

Obtener el nombre y el apellido de los clientes que han comprado como mínimo un coche 'blanco' y un coche 'rojo'.

Solución:

$\{t \mid \exists\ s \in \text{VENTAS}\ (s[color] = \text{'blanco'}$

$\wedge\ \exists\ u \in \text{VENTAS}\ (u[dni] = s[dni] \wedge u[color] = \text{'rojo'}$

$\wedge\ \exists\ v \in \text{CLIENTES}\ (v[dni] = u[dni] \wedge t[nombre] = u[\text{CLIENTES.nombre}]$

$\wedge\ t[apellido] = u[apellido])))\}$

nombre	*apellido*
Luis	García

♦ Problema 3.48

Obtener los valores del atributo *dni* para los clientes que sólo han comprado coches al concesionario con *cifc* igual a '0001'.

Solución:

Este problema se puede plantear como: "obtener los valores del atributo *dni* para los clientes que en la relación VENTAS no les corresponde ningún valor de *cifc* distinto de 0001".

$$\{t \mid \neg\ \exists\ s \in \text{VENTAS}\ (t[dni] = s[dni] \wedge \neg\ (s[cifc] = 0001))\}$$

dni
0002

♦ Problema 3.49

Obtener los valores del atributo *codcoche* de aquellos automóviles que han sido comprados por todos los clientes de 'Madrid'.

Solución:

Para resolver el problema hay que utilizar las relaciones CLIENTES y VENTAS.

$\{t \mid \forall\ s \in \text{CLIENTES}\ ((s[\text{CLIENTES.ciudad}] = \text{'Madrid'})$

$\Rightarrow\ \exists\ u \in \text{VENTAS}\ (u[\text{dni}] = s[\text{dni}] \wedge t[\text{codcoche}] = u[\text{codcoche}]))\}$

Conjunto de todas las tuplas t tal que, para todas las tuplas s de la relación CLIENTES, si el valor de s en el atributo *ciudad* es 'Madrid', entonces existe una tupla u en la relación VENTAS para la que el atributo *dni* toma el mismo valor que aparece en s y t toma el valor del atributo *codcoche* de u.

Por otra parte, como $P_1 \Rightarrow P_2$ equivale a $\neg\ P_1 \vee P_2$, la expresión anterior se puede poner en la forma:

$\{t \mid \forall\ s \in \text{CLIENTES}\ (\neg\ (s[\text{CLIENTES.ciudad}] = \text{'Madrid'})$

$\vee\ \exists\ u \in \text{VENTAS}\ (u[\text{dni}] = s[\text{dni}] \wedge t[\text{codcoche}] = u[\text{codcoche}]))\}$

Conjunto de todas las tuplas t tal que, para todas las tuplas s de la relación CLIENTES, o el valor de s en el atributo *ciudad* es distinto de 'Madrid' o existe una tupla u en la relación VENTAS para la que el atributo *dni* toma el mismo valor que aparece en s y t toma el valor del atributo *codcoche* de u.

codcoche

Capítulo 4
Cálculo Relacional de Dominios

4.1 Introducción

Como se apuntó en el capítulo 3, el *cálculo relacional* proporciona un procedimiento que permite la manipulación de los datos dentro del *modelo relacional*. Mediante el cálculo relacional, se define la relación deseada a partir de las relaciones que componen la base de datos sin dar un método específico para obtenerla.

En el cálculo relacional existen dos enfoques, el *cálculo relacional de tuplas* y el *cálculo relacional de dominios*. El cálculo relacional de dominios difiere del cálculo relacional de tuplas en el empleo de *variables de dominio* en lugar de *variables de tuplas*, es decir, variables que recorren dominios en lugar de variables que recorren relaciones. Las variables de dominio toman valores del dominio de un atributo en lugar de valores de una tupla completa.

En el presente capítulo se proponen y resuelven problemas del cálculo relacional de dominios y se realiza un recordatorio de los aspectos teóricos más importantes en relación con los problemas.

4.2 Cálculo relacional de dominios

En el cálculo relacional de dominios una *consulta* se expresa como el conjunto de todas las tuplas $< x_1, x_2, ..., x_n >$ que verifican el predicado P.

$$\{< x_1, x_2, ..., x_n > \mid P(x_1, x_2, ..., x_n)\}$$

En la expresión anterior, $x_1, x_2, ..., x_n$ representan *variables de dominio* y P representa una *fórmula*.

Una *variable de dominio* x_i es una variable que recorre un dominio D_i, es decir, una variable que tiene como únicos valores permitidos los del dominio D_i.

Por lo tanto, una tupla $< x_1, x_2, ..., x_n >$ estará formada por valores de los dominios recorridos por las variables de dominio $x_1, x_2, ..., x_n$.

Por otra parte, los componentes de una fórmula pueden tener una de las siguientes formas:

- *$< x_1, x_2, ..., x_n >$ pertenece a R*:

  $$< x_1, x_2, ..., x_n > \in R$$

  donde R es una relación con n atributos y $x_1, x_2, ..., x_n$ son variables de dominio. Sean $A_1, A_2, ..., A_n$ los atributos de la relación R y $D_1, D_2, ..., D_n$ los dominios en los que están definidos cada uno de dichos atributos, respectivamente. En la expresión anterior cada variable de dominio x_i puede tomar los valores del dominio D_i restringidos a los valores que toma el atributo A_i en la relación R. La condición de pertenencia resulta verdadera si, y sólo si, existe una tupla en la relación R con el valor de la variable x_1 en el atributo A_1, el valor de la variable x_2 en el atributo A_2, ... y el valor de la variable x_n en el atributo A_n.

- *x_1 operado con x_2* :

  $$x_1 \Theta x_2$$

  donde x_1 y x_2 son variables de dominio y Θ es un operador de comparación ($<, \leq, =, >, \geq$). Los atributos A_1 y A_2 correspondientes a las variables x_1 y x_2 deben pertenecer a dominios cuyos elementos puedan compararse mediante el operador Θ.

- *x_i operado con c*:

  $$x_i \Theta c$$

  donde x_i es una variable de dominio, Θ es un operador de comparación y c es una constante en el dominio del atributo A_i para el que está definida la variable de domino x_i.

Para construir una fórmula, a partir de los componentes anteriores, se deben tener en cuenta las siguientes reglas:

- Cada uno de los componentes descritos es una fórmula.
- Si P_1 es una fórmula, entonces también lo son $\neg P_1$ y (P_1).
- Si P_1 y P_2 son fórmulas, entonces también lo son $P_1 \vee P_2$, $P_1 \wedge P_2$, y $P_1 \Rightarrow P_2$.

- Si $P(x_i)$ es una fórmula en x_i, donde x_i es una variable de dominio, entonces $\exists\, x_i\, (P(x_i))$ y $\forall\, x_i\, (P(x_i))$ también son fórmulas.

La expresión:

$$\exists\, x_i\, (P(x_i))$$

indica que, existe un valor de la variable de dominio x_i para el cual es verdadero el predicado $P(x_i)$.

Como la condición de *existencia* puede estar anidada:

$$\exists\, x_1\, (\exists\, x_2 (\exists\, x_3 \ldots \exists\, x_n\, (P(x_1, x_2, x_3, \ldots, x_n))))$$

para abreviar la notación se puede escribir:

$$\exists\, x_1, x_2, x_3, \ldots, x_n\, (P(x_1, x_2, x_3, \ldots, x_n))$$

Por otra parte, la expresión:

$$\forall\, x_i\, (P(x_i))$$

denota que, el predicado $P(x_i)$ es verdadero para todo valor de la variable de dominio x_i.

Cuando se utilizan las expresiones $P_1 \wedge P_2$, $\forall\, x_i\, (P(x_i))$ y $P_1 \Rightarrow P_2$, en una fórmula, éstas se pueden cambiar por sus respectivas expresiones equivalentes:

$$\neg\, (\neg\, P_1 \vee \neg\, P_2) \text{ en lugar de } P_1 \wedge P_2$$

$$\neg\, \exists\, x_i\, (\neg\, P(x_i)) \text{ en lugar de } \forall\, x_i\, (P(x_i))$$

$$\neg\, P_1 \vee P_2 \text{ en lugar de } P_1 \Rightarrow P_2$$

4.3 Problemas resueltos

En esta sección se proponen y resuelven problemas utilizando el cálculo relacional de dominios. Mientras no se diga lo contrario, en los problemas se hace referencia a la base de datos AUTOMÓVILES descrita en el apéndice A.

Como en la base de datos AUTOMÓVILES existen los atributos *nombre* y *ciudad* en más de una relación, cuando se haga referencia a dichos atributos se les antepondrá el nombre de la relación en la que se encuentran. Así, a los atributos *nombre* y *ciudad* de la relación CLIENTES se les denotará por *CLIENTES.nombre* y *CLIENTES.ciudad*, respectivamente. Lo mismo se hará con los atributos *nombre* de las relaciones MARCAS, COCHES y CONCESIONARIOS y con los atributos *ciudad* de las relaciones MARCAS y CONCESIONARIOS.

♦ Problema 4.1

Obtener todas las tuplas de la relación CLIENTES.

Solución:

En este caso hay que utilizar la relación CLIENTES que tiene por cabecera (*dni*, *nombre*, *apellido*, *ciudad*).

Sean las variables a, b, c y d definidas en los dominios de los atributos *dni*, *CLIENTES.nombre*, *apellido* y *CLIENTES.ciudad*, respectivamente:

a(*dni*), b(*CLIENTES.nombre*), c(*apellido*), d(*CLIENTES.ciudad*)

el problema se resuelve con la expresión:

$$\{< a, b, c, d > | < a, b, c, d > \in \text{CLIENTES}\}$$

Conjunto de todas las tuplas < a, b, c, d >, formadas por valores de los dominios correspondientes a las variables a, b, c y d, tal que, dichas tuplas están limitadas a las que pertenecen a la relación CLIENTES.

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	Garcia	Madrid
0002	Antonio	López	Valencia
0003	Juan	Martín	Madrid
0004	María	García	Madrid
0005	Javier	González	Barcelona
0006	Ana	López	Barcelona

♦ Problema 4.2

Obtener todas las tuplas de la relación DISTRIBUCION cuyo valor en el atributo *cantidad* sea mayor que 10.

Solución:

En este hay que utilizar la relación DISTRIBUCION que tiene por cabecera (*cifc, codcoche, cantidad*).

Si se consideran las variables a, b y c definidas en los dominios de los atributos *cifc*, *codcoche*, y *cantidad*, respectivamente:

a(*cifc*), b(*codcoche*), c(*cantidad*)

el problema se resuelve con la expresión:

$$\{< a, b, c > | < a, b, c > \in \text{DISTRIBUCION} \wedge c > 10\}$$

Conjunto de todas las tuplas < a, b, c >, formadas por valores de los dominios correspondientes a las variables a, b y c, que pertenecen a la relación DISTRIBUCION y que además toman un valor en el atributo *cantidad* mayor que 10.

cifc	*codcoche*	*cantidad*
0005	0016	20

♦ Problema 4.3

Obtener los valores del atributo *cifc* que aparecen junto a un valor del atributo *cantidad* mayor que 10 en las tuplas de la relación DISTRIBUCION.

Solución:

Este problema es similar al anterior salvo en que sólo se quieren los valores del atributo *cifc*.

- Variables de dominio:

$$a(cifc), b(codcoche), c(cantidad)$$

- Expresión:

$$\{< a > | \exists b, c (< a, b, c > \in \text{DISTRIBUCION} \wedge c > 10)\}$$

Conjunto de todas las tuplas < a > tal que, para cada una de ellas, existe un valor de b y existe otro valor de c para los que la tupla < a, b, c > pertenece a la relación DISTRIBUCION siendo el valor de c mayor que 10.

Al estar la variable a definida en el dominio del atributo *cifc*, la relación que contiene todas las tuplas < a > tendrá como cabecera (*cifc*) y en su cuerpo aparecerán todos los posibles valores que pueda tomar el atributo *cifc*.

Por otra parte, las condiciones < a, b, c > $\in$ DISTRIBUCION y c > 10 restringen las tuplas de la relación anterior a los valores de *cifc* que aparecen en la relación DISTRIBUCION junto a un valor el atributo *cantidad* mayor que 10.

cifc
0005

♦ Problema 4.4

Para el concesionario con *cifc* igual a '0005', obtener los valores del atributo *codcoche* que en las tuplas de la relación DISTRIBUCION aparecen junto a un valor del atributo *cantidad* mayor que 10.

Solución:

Este problema difiere del anterior en que en lugar de los valores del atributo *cifc* se quieren los del atributo *codcoche* imponiéndose además una nueva condición sobre *cifc*.

- Variables de dominio:

$$a(cifc),\ b(codcoche),\ c(cantidad)$$

- Expresión:

$$\{< b > \mid \exists\ a, c\ (< a, b, c > \in \text{DISTRIBUCION} \wedge c > 10 \wedge a = 0005)\}$$

Conjunto de todas las tuplas < b > tal que, para cada una de ellas, existe un valor de a y existe otro valor de c para los que la tupla < a, b, c > pertenece a la relación DISTRIBUCION siendo el valor de c mayor que 10 y el valor de a igual a '0005'.

Al estar la variable b definida en el dominio del atributo *codcoche*, la relación que contiene todas las tuplas < b > tendrá como cabecera (*codcoche*) y en su cuerpo aparecerán todos los posibles valores que pueda tomar el atributo *codcoche*.

Por otra parte, las condiciones $< a, b, c > \in$ DISTRIBUCION, $c > 10$ y $a = 0005$ restringen las tuplas de la relación anterior a los valores de *codcoche* que aparecen en la relación DISTRIBUCION junto a un valor el atributo *cifc* igual a '0005' y junto a un valor en atributo *cantidad* mayor que 10.

codcoche
0016

♦ Problema 4.5

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo'.

Solución:

- Relaciones:

VENTAS (*cifc*, *dni*, *codcoche*, *color*)
CLIENTES (*dni*, *nombre*, *apellido*, *ciudad*)

- Variables de dominio:

a(*cifc*), b(*dni*), c(*codcoche*), d(*color*),
e(*CLIENTES. nombre*), f(*apellido*), g(*CLIENTES.ciudad*)

- Expresión:

{< b, g > | ∃ a, c, d (< a, b, c, d > ∈ VENTAS ∧ d = 'rojo')

∧ ∃ e, f (< b, e, f, g > ∈ CLIENTES)}

Conjunto de todas las tuplas < b, g > tal que, para cada una de ellas:

- Existen valores de a, c y d para los que la tupla < a, b, c, d > pertenece a la relación VENTAS siendo el valor de d igual a 'rojo'.
- Y existen valores de e y f para los que la tupla < b, e, f, g > pertenece a la relación CLIENTES.

Al estar la variable b definida en el dominio del atributo *dni* y la variable g definida en el dominio del atributo *CLIENTES.ciudad*, la relación que contiene todas las tuplas < b, g > tendrá como cabecera (*dni, ciudad*) y en su cuerpo aparecerán todas las posibles combinaciones de valores que puedan tomar los atributos *dni* y *CLIENTES.ciudad*.

Por otra parte:

- ∃ a, c, d (< a, b, c, d > ∈ VENTAS ∧ d = 'rojo') restringe las tuplas anteriores < b, g > a las que contienen valores de *dni* que en la relación VENTAS aparecen junto a un valor en el atributo *color* igual a 'rojo'.
- ∃ e, f (< b, e, f, g > ∈ CLIENTES) restringe las tuplas < b, g > a los valores de *dni* y *ciudad* que aparecen en una misma tupla en la relación CLIENTES.

dni	*ciudad*
0001	Madrid
0002	Valencia
0004	Madrid

♦ Problema 4.6

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo' o de color 'blanco' o de ambos colores.

Solución:

El problema es similar al anterior salvo en que el color del coche puede ser 'rojo' o 'blanco'. Para imponer esta condición, en la expresión del problema anterior, se ha sustituido d = 'rojo' por (d = 'rojo' ∨ d = 'blanco').

- Variables de dominio:

  a(*cifc*), b(*dni*), c(*codcoche*), d(*color*),
  e(*CLIENTES. nombre*), f(*apellido*), g(*CLIENTES.ciudad*)

- Expresión:

  {< b, g > | ∃ a, c, d (< a, b, c, d > ∈ VENTAS ∧ (d = 'rojo' ∨ d = 'blanco'))

  ∧ ∃ e, f (< b, e, f, g > ∈ CLIENTES)}

dni	*ciudad*
0001	Madrid
0002	Valencia
0003	Madrid
0004	Madrid

♦ Problema 4.7

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado al menos un coche de color 'rojo' y otro coche de color 'blanco'.

Solución:

Este problema difiere del anterior en que los clientes deben haber adquirido al menos un coche 'rojo' y otro 'blanco'.

En principio se puede pensar que la expresión que resuelve el problema es la misma que la indicada en el problema anterior sustituyendo (d = 'rojo' ∨ d = 'blanco') por (d = 'rojo' ∧ d = 'blanco'), es decir, cambiando ∨ ('o') por ∧ ('y'). En tal caso el resultado sería una relación vacía ya que no es posible que el color sea simultáneamente 'rojo' y 'blanco' en una misma tupla. Para solucionar el problema se debe imponer que en una tupla de la relación

VENTAS el valor del atributo *color* sea 'rojo' y en otra tupla de la relación VENTAS, para el mismo valor de *dni*, el valor del atributo *color* sea 'blanco'.

- Variables de dominio:

a(*cifc*), b(*dni*), c(*codcoche*), d(*color*),
aa(*cifc*), cc(*codcoche*), dd(*color*),
e(*CLIENTES. nombre*), f(*apellido*), g(*CLIENTES.ciudad*)

- Expresión:

{< b, g > | ∃ a, c, d (< a, b, c, d > ∈ VENTAS ∧ d = 'rojo')

∧ ∃ aa, cc, dd (< aa, b, cc, dd > ∈ VENTAS ∧ dd = 'blanco')

∧ ∃ e, f (< b, e, f, g > ∈ CLIENTES)}

Conjunto de todas las tuplas < b, g > tal que, para cada una de ellas:

- Existen valores de a, c y d para los que la tupla < a, b, c, d > pertenece a la relación VENTAS siendo el valor de d igual a 'rojo'.
- Y existen valores de aa, cc y dd para los que la tupla < aa, b, cc, dd > pertenece a la relación VENTAS siendo el valor de dd igual a 'blanco'.
- Y existen valores de e y f para los que la tupla < b, e, f, g > pertenece a la relación CLIENTES.

Al estar la variable b definida en el dominio del atributo *dni* y la variable g definida en el dominio del atributo *ciudad*, la relación que contiene todas las tuplas < b, g > tendrá como cabecera (*dni, ciudad*) y en su cuerpo aparecerán todas las posibles combinaciones de valores que puedan tomar los atributos *dni* y *CLIENTES.ciudad*.

Por otra parte:

- ∃ a, c, d (< a, b, c, d > ∈ VENTAS ∧ d = 'rojo') restringe las tuplas anteriores < b, g > a las que contienen valores de *dni* que aparecen en la relación VENTAS junto a un valor el atributo *color* igual a 'rojo'.
- ∃ aa, cc, dc (< aa, b, cc, dd > ∈ VENTAS ∧ dd = 'blanco') restringe las tuplas < b, g > a las que contienen valores de *dni* que aparecen en la relación VENTAS junto a un valor el atributo *color* igual a 'blanco'.
- ∃ e, f (< b, e, f, g > ∈ CLIENTES) restringe las tuplas < b, g > a los valores de *dni* y *ciudad* que aparecen en una misma tupla en la relación CLIENTES.

dni	*ciudad*
0001	Madrid

♦ Problema 4.8

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo' y alguno que no sea de color 'blanco'.

Solución:

El problema es similar al anterior imponiendo la condición de que los clientes deben haber adquirido al menos un coche 'rojo' y otro que no sea 'blanco'. Para ello, en la expresión del problema anterior, se sustituye la condición dd = 'blanco' por su negación ¬ (dd = 'blanco').

- Variables de dominio:

a(*cifc*), b(*dni*), c(*codcoche*), d(*color*),
aa(*cifc*), cc(*codcoche*), dd(*color*),
e(*CLIENTES. nombre*), f(*apellido*), g(*CLIENTES.ciudad*)

- Expresión:

{< b, g > | ∃ a, c, d (< a, b, c, d > ∈ VENTAS ∧ d = 'rojo')

∧ ∃ aa, cc, dd (< aa, b, cc, dd > ∈ VENTAS ∧ ¬ (dd = 'blanco'))

∧ ∃ e, f (< b, e, f, g > ∈ CLIENTES)}

Obsérvese que para un cliente que hubiera adquirido tres coches de colores 'rojo', 'verde' y 'blanco' en la relación resultante aparecerían sus valores de *dni* y *ciudad*.

dni	*ciudad*

♦ Problema 4.9

Obtener los valores del atributo *dni* para los clientes que han comprado coches de color 'rojo' pero no han comprado coches de color 'blanco'.

Solución:

El problema es similar a los anteriores imponiendo la condición de que en la relación resultante deben aparecer los valores del atributo *dni* de aquellos clientes que habiendo comprado al menos un coche de color 'rojo' no han comprado ninguno de color 'blanco'. Para ello, en la expresión del problema 4.7 se sustituye $\exists$ aa, cc, dd (...) por $\neg \exists$ aa, cc, dd (...) y se elimina $\exists$ e, f (...) por no ser necesario utilizar la relación CLIENTES.

- Variables de dominio:

  a(*cifc*), b(*dni*), c(*codcoche*), d(*color*),
  aa(*cifc*), cc(*codcoche*), dd(*color*)

- Expresión:

  $\{< b > | \exists a, c, d (< a, b, c, d > \in VENTAS \wedge d = 'rojo')$

  $\wedge \neg \exists aa, cc, dd (< aa, b, cc, dd > \in VENTAS \wedge dd = 'blanco')\}$

Conjunto de todas las tuplas < b > tal que, para cada una de ellas:

- Existen valores de a, c y d para los que la tupla < a, b, c, d > pertenece a la relación VENTAS siendo el valor de d igual a 'rojo'.
- Y no existen valores de aa, cc y dd para los que la tupla < aa, b, cc, dd > pertenece a la relación VENTAS con el valor de dd igual a 'blanco'.

Teniendo en cuenta que:

$$\neg \exists aa, cc, dd (< aa, b, cc, dd > \in VENTAS \wedge dd = 'blanco')$$

equivale a:

$$\forall aa, cc, dd (< aa, b, cc, dd > \in VENTAS \Rightarrow \neg (dd = 'blanco'))$$

la expresión anterior se puede poner en la forma:

$\{< b > | \exists a, c, d (< a, b, c, d > \in VENTAS \wedge d = 'rojo')$

$\wedge \forall aa, cc, dd (< aa, b, cc, dd > \in VENTAS \Rightarrow \neg (dd = 'blanco'))\}$

Conjunto de todas las tuplas < b > tal que, para cada una de ellas:

- Existen valores de a, c y d para los que la tupla < a, b, c, d > pertenece a la relación VENTAS siendo el valor de d igual a 'rojo'.
- Y para todos los valores de aa, cc y dd, si la tupla < aa, b, cc, dd > pertenece a la relación VENTAS entonces dd es distinto de 'blanco'.

Al estar la variable b definida en el dominio del atributo *dni*, la relación que contiene todas las tuplas < b > tendrá como cabecera (*dni*) y en su cuerpo aparecerán todos los posibles valores que pueda tomar el atributos *dni*.

Por otra parte:

- $\exists$ a, c, d (< a, b, c, d > $\in$ VENTAS $\wedge$ d = 'rojo') restringen las tuplas anteriores < b > a las que contienen valores de *dni* que aparecen en la relación VENTAS junto a un valor el atributo *color* igual a 'rojo'.
- $\neg \exists$ aa, cc, dd (< aa, b, cc, dd > $\in$ VENTAS $\wedge$ dd = 'blanco'), o su expresión equivalente, restringe las tuplas < b > a los valores de *dni* que no aparecen en ninguna tupla de la relación VENTAS junto a un valor el atributo *color* igual a 'blanco'.

Obsérvese que para un cliente que hubiera adquirido tres coches de colores 'rojo', 'verde' y 'blanco' en la relación resultante no aparecería su *dni*.

dni
0002
0004

♦ Problema 4.10

Obtener los valores del atributo *dni* para los clientes que no han comprado coches de color 'rojo' o han comprado al menos un coche de color 'blanco'.

Solución:

- Relación:

VENTAS (*cifc*, *dni*, *codcoche*, *color*)

- Variables de dominio:

a(*cifc*), b(*dni*), c(*codcoche*), d(*color*),
aa(*cifc*), b(*dni*), cc(*codcoche*), dd(*color*)

- Expresión:

{< b > | $\neg \exists$ a, c, d (< a, b, c, d > $\in$ VENTAS $\wedge$ d = 'rojo')

$\vee \exists$ aa, cc, dd (< aa, b, cc, dd > $\in$ VENTAS $\wedge$ d = 'blanco')}

Conjunto de todas las tuplas < b > tal que, para cada una de ellas:

- No existen valores de a, c y d para los que la tupla < a, b, c, d > pertenece a la relación VENTAS siendo el valor de d igual a 'rojo'.
- O existen valores de aa, cc y dd para los que la tupla < aa, b, cc, dd > pertenece a la relación VENTAS con el valor de dd igual a 'blanco'.

Obsérvese que la condición "no han comprado coches de color 'rojo' o han comprado al menos un coche de color 'blanco' ($\neg P_1 \vee P_2$)" es equivalente a la condición "si han comprado coches de color 'rojo' entonces han comprado al menos un coche de color 'blanco' ($P_1 \Rightarrow P_2$)". Por lo que la expresión anterior se puede poner en la forma:

$\{< b > \mid \exists a, c, d\ (< a, b, c, d > \in \text{VENTAS} \wedge d = \text{'rojo'})$

$\Rightarrow \exists aa, cc, dd\ (< aa, b, cc, dd > \in \text{VENTAS} \wedge d = \text{'blanco'})\}$

dni
0001
0003
0005

♦ Problema 4.11

Obtener el nombre de los clientes que han comprado coches en todos los concesionarios.

Solución:

- Relaciones:

CONCESIONARIOS (*cifc*, *nombre*, *ciudad*)
VENTAS (*cifc*, *dni*, *codcoche*, *color*)
CLIENTES (*dni*, *nombre*, *apellido*, *ciudad*)

- Variables de dominio:

a(*cifc*), b(*CONCESIONARIOS.nombre*), c(*CONCESIONARIOS.ciudad*),
d(*dni*), e(*codcoche*), f(*color*),
g(*CLIENTES.nombre*), h(*apellido*), i(*CLIENTES.ciudad*)

- Expresión:

$\{< g > \mid \forall a, b, c\ ((< a, b, c > \in \text{CONCESIONARIOS})$

$\Rightarrow (\exists d, e, f\ (< a, d, e, f > \in \text{VENTAS} \wedge \exists h, i\ (< d, g, h, i > \in \text{CLIENTES})))\}$

Conjunto de todas las tuplas < g > tal que, para todos los valores de a, b y c:

- Si la tupla < a, b, c > pertenece a la relación CONCESIONARIOS.
- Entonces, existen valores de d, e y f para los que la tupla < a, d, e, f > pertenece a la relación VENTAS y existen valores de h e i para los que la tupla < d, g, h, i > pertenece a la relación CLIENTES.

Al ser $P_1 \Rightarrow P_2$ equivalente a $\neg P_1 \vee P_2$, la expresión anterior se puede poner como:

$\{< g > \mid \forall a, b, c (\neg (< a, b, c > \in CONCESIONARIOS)$

$\vee (\exists d, e, f (< a, d, e, f > \in VENTAS \wedge \exists h, i (< d, g, h, i > \in CLIENTES)))\}$

Conjunto de todas las tuplas < g > tal que, para todos los valores de a, b y c:

- O < a, b, c > no pertenece a la relación CONCESIONARIOS.
- O existen valores de d, e y f para los que la tupla < a, d, e, f > pertenece a la relación VENTAS y existen valores de las variables h e i para los que la tupla < d, g, h, i > pertenece a la relación CLIENTES.

nombre

♦ Problema 4.12

Sean COCHES1 y COCHES2 dos relaciones en las que aparecen las tuplas de la relación COCHES correspondientes al modelo 'gti' y las tuplas de la relación COCHES que tienen por nombre 'ibiza', respectivamente. Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional COCHES1 ∪ COCHES2.

COCHES1

codcoche	*nombre*	*modelo*
0002	ibiza	gti
0005	cordoba	gti
0007	megane	gti
0016	Astra	gti

COCHES2

codcoche	*nombre*	*modelo*
0001	ibiza	glx
0002	ibiza	gti
0003	ibiza	gtd

Solución:

- Variables de dominio:

a(*codcoche*), b(*COCHES.nombre*), c(*mdelo*)

- Expresión:

$$\{< a, b, c > | < a, b, c > \in COCHES1 \vee < a, b, c > \in COCHES2\}$$

Conjunto de todas las tuplas < a, b, c > tal que cada una de ellas pertenece a la relación COCHES1 o pertenece a la relación COCHES2.

codcoche	*nombre*	*modelo*
0002	ibiza	gti
0005	cordoba	gti
0007	megane	gti
0016	astra	gti
0001	ibiza	glx
0003	ibiza	gtd

♦ Problema 4.13

Sean COCHES1 y COCHES2 las relaciones indicadas en el problema 3.12. Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional COCHES1 $\cap$ COCHES2.

Solución:

- Variables de dominio:

a(*codcoche*), b(*COCHES.nombre*), c(*mdelo*)

- Expresión:

$$\{< a, b, c > | < a, b, c > \in COCHES1 \wedge < a, b, c > \in COCHES2\}$$

Conjunto de todas las tuplas < a, b, c > tal que cada una de ellas pertenece a la relación COCHES1 y a la relación COCHES2.

codcoche	*nombre*	*modelo*
0002	ibiza	gti

♦ Problema 4.14

Sean COCHES1 y COCHES2 las relaciones indicadas en el problema 3.12. Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional COCHES1 - COCHES2.

Solución:

- Variables de dominio:

  a(*codcoche*), b(*COCHES.nombre*), c(*mdelo*)

- Expresión:

  $\{< a, b, c > | < a, b, c > \in COCHES1 \wedge \neg < a, b, c > \in COCHES2\}$

Conjunto de todas las tuplas < a, b, c > tal que cada una de ellas pertenece a la relación COCHES1 pero no pertenece a la relación COCHES2.

codcoche	*nombre*	*modelo*
0005	cordoba	gti
0007	megane	gti
0016	astra	gti

♦ Problema 4.15

Sean NMARCAS y NCONCESION dos relaciones en las que aparecen los nombres de las marcas de coches y los nombres de los concesionarios, respectivamente. Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional producto cartesiano NMARCAS × NCONCESION.

NMARCAS

mnombre
seat
renault
citroen
audi
opel
bmw

NCONCESION

cnombre
acar
bcar
ccar
dcar
ecar

Solución:

- Variables de dominio:

  a(*mnombre*), b(*cnombre*)

- Expresión:

  $\{< a, b > | < a > \in NMARCAS \wedge < b > \in NCONCESION\}$

Conjunto de todas las tuplas < a, b > tal que, para cada una de ellas:

- Existe una tupla < a > en la relación NMARCAS.
- Y existe otra tupla < b > en la relación NCONCESION.

mnombre	*cnombre*
seat	acar
seat	bcar
seat	ccar
seat	dcar
seat	ecar
renault	acar
renault	bcar
renault	ccar
renault	dcar
renault	ecar

(continuación)

mnombre	*cnombre*
citroen	acar
citroen	bcar
citroen	ccar
citroen	dcar
citroen	ecar
audi	acar
audi	bcar
audi	ccar
audi	dcar
audi	ecar

(continuación)

mnombre	*cnombre*
opel	acar
opel	bcar
opel	ccar
opel	dcar
opel	ecar
bmw	acar
bmw	bcar
bmw	ccar
bmw	dcar
bmw	ecar

♦ Problema 4.16

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional $\sigma_{\text{ciudad = 'Barcelona'}}$ (MARCAS).

Solución:

- Variables de dominio:

  a(*cifn*), b(*MARCA.nombre*), c(*MARCA.ciudad*)

- Expresión:

  $\{< a, b, c > \mid < a, b, c > \in \text{MARCAS} \wedge c = \text{'Barcelona'}\}$

Conjunto todas las tuplas < a, b, c > que verifican la condición de pertenecer a la relación MARCAS y que además verifican la condición c = 'Barcelona'.

cifm	*nombre*	*ciudad*
0002	renault	Barcelona
0006	bmw	Barcelona

♦ Problema 4.17

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional $\sigma_{\text{apellido = 'García' AND ciudad = 'Madrid'}}$ (CLIENTES).

Solución:

- Variables de dominio:

  a(*dni*), b(*CLIENTES.nombre*), c(*apellido*), d(*CLIENTES.ciudad*)

- Expresión:

  $\{< a, b, c, d > \mid < a, b, c, d > \in \text{CLIENTES} \wedge c = \text{'García'} \wedge d = \text{'Madrid'}\}$

Conjunto todas las tuplas < a, b, c, d > que verifican la condición de pertenecer a la relación CLIENTES y que además toman valores c = 'García' y d = 'Madrid'.

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	García	Madrid
0004	María	García	Madrid

♦ Problema 4.18

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional Π_{apellido} (CLIENTES).

Solución:

- Variables de dominio:

  a(*dni*), b(*CLIENTES.nombre*), c(*apellido*), d(*CLIENTES.ciudad*)

- Expresión:

  $$\{< c > \mid \exists\, a, b, d\ (< a, b, c, d > \in \text{CLIENTES})\}$$

Conjunto de todas las tuplas < c > tal que, para cada una de ellas, existen valores de a, b y d para los que la tupla < a, b, c, d > pertenece a la relación CLIENTES.

apellido
García
López
Martín
González

♦ Problema 4.19

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional $\Pi_{\text{apellido, cuidad}}$ (CLIENTES).

Solución:

- Variables de dominio:

  a(*dni*), b(*CLIENTES.nombre*), c(*apellido*), d(*CLIENTES.ciudad*)

- Expresión:

  $$\{< c, d > \mid \exists\, a, b\ (< a, b, c, d > \in \text{CLIENTES})\}$$

Conjunto de todas las tuplas < c, d > tal que, para cada una de ellas, existen valores de a y b para los que la tupla < a, b, c, d > pertenece a la relación CLIENTES.

apellido	*ciudad*
Garcia	Madrid
López	Valencia
Martín	Madrid
González	Barcelona
López	Barcelona

♦ Problema 4.20

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional $\Pi_{\text{apellido, ciudad}}$ ($\sigma_{\text{ciudad = 'Madrid'}}$ (CLIENTES)).

Solución:

- Variables de dominio:

  a(*dni*), b(*CLIENTES.nombre*), c(*apellido*), d(*CLIENTES.ciudad*)

- Expresión:

  $$\{< c, d > \mid \exists\, a, b\ (< a, b, c, d > \in \text{CLIENTES} \wedge d = \text{'Madrid'})\}$$

Conjunto de todas las tuplas < c, d > tal que, para cada una de ellas, existen valores de a y b para los que la tupla < a, b, c, d > pertenece a la relación CLIENTES siendo d = 'Madrid'.

apellido	ciudad
García	Madrid
Martín	Madrid

♦ Problema 4.21

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional (MARCAS $\rho_{mnombre}$ (NOMBRE)) * CLIENTES (reunión natural de las relaciones MARCAS y CLIENTES según el atributo *ciudad*).

Solución:

- Relaciones:

MARCAS (*cifm, nombre, ciudad*)
CLIENTES (*dni, nombre, apellido, ciudad*)

- Variables de dominio:

a(*cifm*), b(*MARCAS.nombre*), c(*MARCAS.ciudad*),
d(*dni*), e(*CLIENTES. nombre*), f(*apellido*), g(*CLIENTES.ciudad*)

- Expresión:

$\{< a, b, c, d, e, f > | < a, b, c > \in \text{MARCAS}$

$\wedge \exists g\ (< d, e, f, g > \in \text{CLIENTES} \wedge g = c)\}$

Obsérvese que la condición g = c es la que obliga a que la reunión se realice por el atributo común *ciudad* y que en la relación resultante el atributo *MARCAS.nombre* se ha denotado como *mnombre*.

cifm	mnombre	dni	nombre	apellido	ciudad
0001	seat	0001	Luis	García	Madrid
0001	seat	0003	Juan	Martín	Madrid
0001	seat	0004	María	García	Madrid
0002	renault	0005	Javier	González	Barcelona
0002	renault	0006	Ana	López	Barcelona
0003	citroen	0002	Antonio	López	Valencia
0004	audi	0001	Luis	García	Madrid
0004	audi	0003	Juan	Martín	Madrid
0004	audi	0004	María	García	Madrid
0006	bmw	0005	Javier	González	Barcelona
0006	bmw	0006	Ana	López	Barcelona

♦ Problema 4.22

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional (MARCAS $\rho_{mnombre,\ mciudad}$ (nombre, ciudad)) $|\times|_{mciudad\ >\ ciudad}$ CLIENTES (reunión 'mayor que' de la relación MARCAS según el atributo *ciudad* con la relación CLIENTES según el atributo *ciudad*).

Solución:

- Relaciones:

MARCAS (*cifm*, *nombre*, *ciudad*)
CLIENTES (*dni*, *nombre*, *apellido*, *ciudad*)

- Variables de dominio:

a(*cifm*), b(*MARCAS.nombre*), c(*MARCAS.ciudad*),
d(*dni*), e(*CLIENTES. nombre*), f(*apellido*), g(*CLIENTES.ciudad*)

- Expresión:

$\{< a, b, c, d, e, f, g > | < a, b, c > \in$ MARCAS

$\wedge < d, e, f, g > \in$ CLIENTES $\wedge\ c > g\}$

Obsérvese que este problema se resuelve de forma similar al anterior cambiando la condición g = c por c > g y haciendo que aparezcan en la relación resultante las ciudades de las marcas y de los clientes. En la relación resultante los atributos *MARCAS.nombre* y *MARCAS.ciudad* se ha denotado como *mnombre* y *mciudad*, respectivamente.

cifm	*mnombre*	*mciudad*	*dni*	*nombre*	*apellido*	*ciudad*
0001	seat	Madrid	0005	Javier	González	Barcelona
0001	seat	Madrid	0006	Ana	López	Barcelona
0003	citroen	Valencia	0001	Luis	García	Madrid
0003	citroen	Valencia	0003	Juan	Martín	Madrid
0003	citroen	Valencia	0004	María	García	Madrid
0003	citroen	Valencia	0005	Javier	González	Barcelona
0003	citroen	Valencia	0006	Ana	López	Barcelona
0004	audi	Madrid	0005	Javier	González	Barcelona
0004	audi	Madrid	0006	Ana	López	Barcelona
0005	opel	Bilbao	0005	Javier	González	Barcelona
0005	opel	Bilbao	0006	Ana	López	Barcelona

♦ Problema 4.23

Indicar una expresión del cálculo relacional de tuplas equivalente a la expresión del álgebra relacional DIVIDENDO ÷ DIVISOR.

DIVIDENDO

atributo1	*atributo2*
dato1	dato6
dato1	dato8
dato1	dato9
dato2	dato6
dato3	dato5
dato3	dato6
dato4	dato10
dato4	dato9
dato4	dato8
dato4	dato7
dato4	dato6
dato4	dato5

DIVISOR

atributo2
dato6
dato8

Solución:

- Relaciones:

DIVIDENDO (*atributo1*, *atributo2*)
DIVISOR (*atributo2*)

- Variables de dominio:

a(*atributo1*), b(*atributo2*)

- Expresión:

$$\{< a > \mid \forall\ b\ ((< b > \in \text{DIVISOR}) \Rightarrow (<a, b> \in \text{DIVIDENDO}))\}$$

Conjunto de todas las tuplas < a > tal que, para todas los valores de b, si la tupla < b > pertenece a la relación DIVISOR entonces la tupla < a, b > pertenece a la relación DIVIDENDO.

atributo1
dato1
dato4

♦ Problema 4.24

Obtener los nombres de las marcas que tienen modelos 'gtd'.

Solución:

- Relaciones:

COCHES (*codcoche*, *nombre*, *modelo*)
MARCO (*cifm*, *codcoche*)

MARCAS (*cifm*, *nombre*, *ciudad*)

- Variables de dominio:

a(*codcoche*), b(*COCHES. nombre*), c(*modelo*),
d(*cifm*), e(*MARCAS.nombre*), f(*MARCAS.ciudad*)

- Expresión:

$\{< e > \mid \exists\, a, b, c\ (< a, b, c > \in \text{COCHES} \wedge c = \text{'gtd'}$

$\wedge \exists\, d\ (< d, a > \in \text{MARCO} \wedge \exists\, f\ (< d, e, f > \in \text{MARCA})))\}$

Una forma para resolver los problemas mediante el cálculo relacional es plantear primero su solución en álgebra relacional y considerar las expresiones equivalentes en cálculo relacional.

Obsérvese que en el álgebra relacional el problema se puede resolver con las siguientes operaciones:

- Selección de las tuplas de la relación COCHES en las que el atributo *modelo* toma el valor 'gtd'.

$$\{< a, b, c > \mid < a, b, c > \in \text{COCHES} \wedge c = \text{'gtd'}\}$$

- Reunión natural de la relación anterior con la relación MARCO según el atributo común *codcoche*.

$$\{< a, b, c > \mid < a, b, c > \in \text{COCHES} \wedge c = \text{'gtd'} \wedge \exists\, d\ (< d, a > \in \text{MARCO})\}$$

- Reunión natural de la relación anterior con la relación COCHES según el atributo común *cifm*.

$$\{< a, b, c > \mid < a, b, c > \in \text{COCHES} \wedge c = \text{'gtd'} \wedge \exists\, d\ (< d, a > \in \text{MARCO} \wedge \exists\, e, f\ (< d, e, f > \in \text{MARCA}))\}$$

- Proyección por el atributo *nombre* de la relación MARCAS.

$$\{< e > \mid \exists\, a, b, c\ (< a, b, c > \in \text{COCHES} \wedge c = \text{'gtd'} \wedge \exists\, d\ (< d, a > \in \text{MARCO} \wedge \exists\, f\ (< d, e, f > \in \text{MARCA})))\}$$

nombre
seat
renault
citroen

♦ Problema 4.25

Obtener los nombres de las marcas de las que se han vendido coches de color 'rojo'.

Solución:

- Relaciones:

VENTAS (*cifm*, *dni*, *codcoche*, *color*)
MARCO (*cifm*, *codcoche*)
MARCAS (*cifm*, *nombre*, *ciudad*)

- Variables de dominio:

a(*cifm*), b(*dni*), c(*codcoche*), d(*color*),
e(*cifm*), f(*MARCAS.nombre*), g(*MARCAS.ciudad*)

- Expresión:

$\{< f > | \exists a, b, c, d (< a, b, c, d > \in VENTAS \wedge d = 'rojo'$

$\wedge \exists e (< e, c > \in MARCO \wedge \exists g (< e, f, g > \in MARCAS))\}$

Obsérvese que en el álgebra relacional el problema se puede resolver con las siguientes operaciones:

- Selección de las tuplas de la relación VENTAS en las que el atributo *color* toma el valor 'rojo'.

$\{< a, b, c, d > | < a, b, c, d > \in VENTAS \wedge d = 'rojo'\}$

- Reunión natural de la relación anterior con la relación MARCO según el atributo común *codcoche*.

$\{< a, b, c, d > | < a, b, c, d > \in VENTAS \wedge d = 'rojo'$
$\wedge \exists e (< e, c > \in MARCO)\}$

- Reunión natural de la relación anterior con la relación COCHES según el atributo común *cifm*.

$\{< a, b, c, d > | < a, b, c, d > \in VENTAS \wedge d = 'rojo'$
$\wedge \exists e (< e, c > \in MARCO$
$\wedge \exists f, g (< e, f, g > \in MARCAS))\}$

- Proyección por el atributo *nombre* de la relación MARCAS.

$\{< f > | \exists a, b, c, d (< a, b, c, d > \in VENTAS \wedge d = 'rojo'$
$\wedge \exists e (< e, c > \in MARCO$

$\wedge \exists g (< e, f, g > \in MARCAS))\}$

nombre
seat
renault
citroen

♦ **Problema 4.26**

Obtener los nombres de los coches que tengan al menos todos los modelos que tiene el coche de nombre 'cordoba'.

Solución:

- Relación:

  COCHES (*codcoche*, *nombre*, *modelo*)

- Variables de dominio:

  a(*codcoche*), b(*COCHES.nombre*), c(*modelo*),
  e(*codcoche*), f(*COCHES.nombre*), g(*modelo*)

- Expresión:

  $\{< f > | \forall a, b, c ((< a, b, c > \in COCHES \wedge b = 'cordoba')$

  $\Rightarrow \exists e, g (< e, f, g > \in COCHES \wedge g = c))\}$

  Como $P_1 \Rightarrow P_2$ equivale a $\neg P_1 \vee P_2$, una expresión equivalente es:

  $\{< f > | \forall a, b, c (\neg (< a, b, c > \in COCHES \wedge b = 'cordoba')$

  $\vee \exists e, g (< e, f, g > \in COCHES \wedge g = c))\}$

  Por otra parte, como $\forall x_i (P(x_i))$ equivale a $\neg \exists x_i (\neg P(x_i))$ y $\neg (\neg P_1 \vee \neg P_2)$ equivale a $P_1 \wedge P_2$, otra expresión equivalente es:

  $\{< f > | \neg \exists a, b, c ((< a, b, c > \in COCHES \wedge b = 'cordoba')$

  $\wedge \neg \exists e, g (< e, f, g > \in COCHES \wedge g = c))\}$

En álgebra relacional el problema se resuelve mediante la división de las relaciones DIVIDENDO entre DIVISOR, siendo DIVIDENDO la relación resultante de proyectar la relación COCHES, según los atributos *nombre* y

modelo, y DIVISOR la relación que resulta al proyectar, según el atributo *modelo*, las tuplas de la relación COCHES para las que el atributo *nombre* es igual a 'cordoba'.

nombre
ibiza
cordoba
megane
astra

♦ Problema 4.27

Obtener el nombre de los coches que no tengan modelo 'gtd'.

Solución:

- Relación:

  COCHES (*codcoche*, *nombre*, *modelo*)

- Variables de dominio:

  a(*codcoche*), b(*COCHES.nombre*), c(*modelo*)

- Expresión:

  $\{< b > | \neg \exists a, c (< a, b, c > \in COCHES \wedge c = 'gtd')\}$

nombre
cordoba
megane
zx
a4
astra
corsa
300
500
700

Obsérvese que el problema se han replanteado como, "obtener el nombre de los coches para los que no existe ninguna tupla en la relación COCHES en la que el atributo *modelo* toma el valor gtd".

♦ Problema 4.28

Obtener todas las tuplas de la relación CONCESIONARIOS.

Solución:

- Relación:

CONCESIONARIOS (*cifc*, *nombre*, *ciudad*)

- Variables de dominio:

a(*cifc*), b(*CONCESIONARIOS.nombre*), c(*CONCESIONARIOS.ciudad*)

- Expresión:

$\{< a, b, c > \mid < a, b, c > \in \text{CONCESIONARIOS}\}$

cifc	*nombre*	*ciudad*
0001	acar	Madrid
0002	bcar	Madrid
0003	ccar	Barcelona
0004	dcar	Valencia
0005	ecar	Bilbao

♦ Problema 4.29

Obtener todas las tuplas de todos los clientes de 'Madrid'.

Solución:

- Relación:

CLIENTES (*dni*, *nombre*, *apellido*, *ciudad*)

- Variables de dominio:

a(*dni*), b(*CLIENTES.nombre*), c(*apellidos*), d(*CLIENTES.ciudad*)

- Expresión:

$\{< a, b, c, d > \mid < a, b, c, d > \in \text{CLIENTES} \wedge d = \text{'Madrid'}\}$

En álgebra relacional esta expresión equivale a seleccionar las tuplas de la relación CLIENTES cuyo valor en el atributo *ciudad* es 'Madrid'.

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	García	Madrid
0003	Juan	Martín	Madrid
0004	María	García	Madrid

♦ Problema 4.30

Obtener todas las parejas de atributos *cifm* de la relación MARCAS y *dni* de la relación CLIENTES que sean de la misma ciudad.

Solución:

- Relaciones:

MARCAS (*cifm*, *nombre*, *ciudad*)
CLIENTES (*dni*, *nombre*, *apellido*, *ciudad*)

- Variables de dominio:

a(*cifm*), b(*MARCAS.nombre*), c(*MARCAS.ciudad*),
d(*dni*), e(*CLIENTES.nombre*), f(*apellidos*), g(*CLIENTES.ciudad*)

- Expresión:

$\{< a, d > \mid \exists\, b, c\ (< a, b, c > \in \text{MARCAS}$

$\wedge \exists\, e, f, g\ (< d, e, f, g > \in \text{CLIENTES} \wedge g = c))\}$

cifm	*dni*
0001	0001
0001	0003
0001	0004
0002	0005
0002	0006
0003	0002
0004	0001
0004	0003
0004	0004
0006	0005
0006	0006

En álgebra relacional esta expresión equivale a la reunión natural de las relaciones MARCAS y CLIENTES, según el atributo común *ciudad*, proyectada según los atributos *cifm* y *dni*.

♦ Problema 4.31

Obtener todas las parejas de valores de los atributos *cifm* de la relación MARCAS y *dni* de la relación CLIENTES que no sean de la misma ciudad.

Solución:

- Relaciones:

MARCAS (*cifm*, *nombre*, *ciudad*)
CLIENTES (*dni*, *nombre*, *apellido*, *ciudad*)

- Variables de dominio:

a(*cifm*), b(*MARCAS.nombre*), c(*MARCAS.ciudad*),
d(*dni*), e(*CLIENTES.nombre*), f(*apellidos*), g(*CLIENTES.ciudad*)

- Expresión:

$\{< a, d > \mid \exists b, c\ (< a, b, c > \in \text{MARCAS}$

$\wedge \exists e, f, g\ (< d, e, f, g > \in \text{CLIENTES} \wedge \neg (g = c)))\}$

En álgebra relacional esta expresión equivale a la reunión *'distinto que'* de las relaciones MARCAS y CLIENTES, para el atributo común *ciudad*, proyectada según los atributos *cifm* y *dni*.

cifm	*dni*
0001	0002
0001	0005
0001	0006
0002	0001
0002	0002
0002	0003
0002	0004
0003	0001
0003	0003
0003	0004
0003	0005
0003	0006
0004	0002

(continuación)

cifm	*dni*
0004	0005
0004	0006
0005	0001
0005	0002
0005	0003
0005	0004
0005	0005
0005	0006
0006	0001
0006	0002
0006	0003
0006	0004

♦ Problema 4.32

Obtener los valores del atributo *codcoche* para los coches que se encuentran en algún concesionario de 'Barcelona'.

Solución:

- Relaciones:

CONCESIONARIOS (*cifc*, *nombre*, *ciudad*)
DISTRIBUCION (*cifc*, *codcoche*, *cantidad*)

- Variables de dominio:

a(*cifc*), b(*CONCESIONARIOS.nombre*), c(*CONCESIONARIOS.ciudad*), d(*cifc*), e(*codcoche*), f(*cantidad*)

- Expresión:

$\{< e > \mid \exists\, a, b, c\ (< a, b, c > \in \text{CONCESIONARIOS} \wedge c = \text{'Barcelona'}$

$\wedge \exists\, f\ (< d, e, f > \in \text{DISTRIBUCIÓN} \wedge d = a))\}$

codcoche
0010
0011
0012

♦ Problema 4.33

Obtener el valor del atributo *codcoche* de aquellos coches vendidos a clientes de 'Madrid'.

Solución:

- Relaciones:

CLIENTES (*dni*, *nombre*, *apellido*, *ciudad*)
VENTAS (*cifc*, *dni*, *codcoche*, *color*)

- Variables de dominio:

a(*dni*), b(*CLIENTES.nombre*), c(*apellidos*), d(*CLIENTES.ciudad*), e(*cifc*), f(*codcoche*), g(*color*)

- Expresión:

$\{< f > \mid \exists\, a, b, c, d\ (< a, b, c, d > \in \text{CLIENTES} \wedge d = \text{'Madrid'}$

$\wedge \exists\, e, g\ (< e, a, f, g > \in \text{VENTAS}))\}$

codcoche
0001
0006
0011
0008

♦ Problema 4.34

Obtener los valores del atributo *codcoche* para los coches que han sido adquiridos por un cliente de 'Madrid' en un concesionario de 'Madrid'.

Solución:

- Relaciones:

CLIENTES (*dni*, *nombre*, *apellido*, *ciudad*)
VENTAS (*cifc*, *dni*, *codcoche*, *color*)
CONCESIONARIOS (*cifc*, *nombre*, *ciudad*)

- Variables de dominio:

a(*dni*), b(*CLIENTES.nombre*), c(*apellidos*), d(*CLIENTES.ciudad*),
e(*cifc*), f(*codcoche*), g(*color*),
h(*CONCESIONARIOS.nombre*), i(*CONCESIONARIOS.ciudad*)

- Expresión:

$\{< f > \mid \exists\, a, b, c, d\ (< a, b, c, d > \in \text{CLIENTES} \wedge d = \text{'Madrid'}$

$\wedge \exists\, e, g\ (< e, a, f, g > \in \text{VENTAS}$

$\wedge \exists\, h, i\ (< e, h, i > \in \text{CONCESIONARIOS} \wedge i = \text{'Madrid'}))\}$

codcoche
0001
0008
0006

♦ Problema 4.35

Obtener los valores del atributo *codcoche* para los coches comprados en un concesionario de la misma ciudad que la del cliente que lo compra.

Solución:

El problema se resuelve de forma similar al anterior imponiendo que las ciudades del concesionario y del cliente sean las mismas.

$\{< f > \mid \exists\, a, b, c, d\ (< a, b, c, d > \in \text{CLIENTES}$

$\wedge \exists\, e, g\ (< e, a, f, g > \in \text{VENTAS}$

$\wedge \exists\, h, i\ (< e, h, i > \in \text{CONCESIONARIOS} \wedge i = d))\}$

codcoche
0001
0008
0006

♦ Problema 4.36

Obtener los valores del atributo *codcoche* para los coches comprados en un concesionario de distinta ciudad que la del cliente que lo compra.

Solución:

La forma de resolver este problema es similar a la aplicada en el problema anterior, imponiendo que las ciudades del concesionario y del cliente sean distintas.

$\{< f > \mid \exists\, a, b, c, d\ (< a, b, c, d > \in \text{CLIENTES}$

$\wedge \exists\, e, g\ (< e, a, f, g > \in \text{VENTAS}$

$\wedge \exists\, h, i\ (< e, h, i > \in \text{CONCESIONARIOS} \wedge \neg\ (i = d)))\}$

codcoche
0005
0011
0014

♦ Problema 4.37

Obtener todas las parejas de nombres de marcas que sean de la misma ciudad.

Solución:

- Relación:

MARCAS (*cifm*, *nombre*, *ciudad*)

- Variables de dominio:

a(*cifm*), b(*MARCAS.nombre*), c(*MARCAS.ciudad*)
aa(*cifm*), bb(*MARCAS.nombre*), cc(*MARCAS.ciudad*)

- Expresión:

{< b, bb > | ∃ a, b, c (< a, b, c > ∈ MARCAS

∧ ∃ aa, bb, cc (< aa, bb, cc > ∈ MARCAS ∧ cc = c ∧ bb < b))}

La condición bb < b permite eliminar las parejas (nombre1, nombre1) de nombres iguales y garantizar la aparición de una sola de las parejas (nombre1, nombre2) y (nombre2, nombre1).

nombre1	*nombre2*
audi	seat
bmw	renault

♦ Problema 4.38

Obtener las parejas de modelos de coches cuyo nombre es el mismo y cuya marca es de 'Bilbao'.

Solución:

- Relaciones:

MARCAS (*cifm*, *nombre*, *ciudad*)
MARCO (*cifm*, *codcoche*)
COCHES (*codcoche*, *nombre*, *modelo*)

- Variables de dominio:

a(*cifm*), b(*MARCAS.nombre*), c(*MARCAS.ciudad*),
d(*codcoche*), e(*COCHES. nombre*), f(*modelo*),
dd(*codcoche*), ee(*COCHES. nombre*), ff(*modelo*)

- Expresión:

{< f, ff > | ∃ a, b, c (< a, b, c > ∈ MARCAS ∧ c = 'Bilbao'

$\wedge \exists$ d (< a, d > $\in$ MARCO

$\wedge \exists$ e (< d, e, f > $\in$ COCHES

$\wedge \exists$ dd, ee (< dd, e, ff > $\in$ COCHES $\wedge$ ee = e $\wedge$ ff < f))))}

modelo1	*modelo2*
caravan	gti

♦ Problema 4.39

Obtener el color de los coches vendidos por el concesionario 'acar'.

Solución:

- Relaciones:

CONCESIONARIOS (*cifc*, *nombre*, *ciudad*)
VENTAS (*cifc*, *dni*, *codcoche*, *color*)

- Variables de dominio:

a(*cifc*), b(*CONCESIONARIOS.nombre*), c(*CONCESIONARIOS.ciudad*),
d(*dni*), e(*codcoche*), f(*color*)

- Expresión:

{< f > | $\exists$ a, b, c (< a, b, c > $\in$ CONCESIONARIOS $\wedge$ b = 'acar'
$\wedge \exists$ d, e (< a, d, e, f > $\in$ VENTAS))}

color
blanco
rojo

♦ Problema 4.40

Obtener el nombre y el apellido de los clientes que han adquirido un coche modelo 'gti' de color 'blanco'.

Solución:

- Relaciones:

CLIENTES (*dni*, *nombre*, *apellido*, *ciudad*)

VENTAS (*cifc*, *dni*, *codcoche*, *color*)
COCHES (*codcoche*, *nombre*, *modelo*)

- Variables de dominio:

  a(*dni*), b(*CLIENTES.nombre*), c(*apellidos*), d(*CLIENTES.ciudad*),
  e(*cifc*), f(*codcoche*), g(*color*),
  h(*COCHES.nombre*), i(*modelo*)

- Expresión:

  {< b, c > | ∃ a, d (< a, b, c, d > ∈ CLIENTES

  ∧ ∃ e, f, g (< e, a, f, g > ∈ VENTAS ∧ g = 'blanco'

  ∧ ∃ h, i (< f, h, i > ∈ COCHES ∧ i = 'gti'))))}

nombre	*apellido*

♦ Problema 4.41

Obtener el nombre y el apellido de los clientes cuyo *dni* es menor que el correspondiente al cliente de nombre 'Juan' y apellido 'Martín'.

Solución:

- Relación:

  CLIENTES (*dni*, *nombre*, *apellido*, *ciudad*)

- Variables de dominio:

  a(*dni*), b(*CLIENTES.nombre*), c(*apellidos*), d(*CLIENTES.ciudad*),
  aa(*dni*), bb(*CLIENTES.nombre*), cc(*apellidos*), dd(*CLIENTES.ciudad*)

- Expresión:

  {< bb, cc > | ∃ a, b, c, d (< a, b, c, d > ∈ CLIENTES

  ∧ b = 'Juan' ∧ c = 'Martín'

  ∧ ∃ aa, dd (< aa, bb, cc, dd > ∈ CLIENTEs ∧ aa < a))}

nombre	*apellido*
Luis	García
Antonio	López

♦ Problema 4.42

Obtener los valores del atributo *dni* para los clientes que sólo han comprado coches al concesionario con *cifc* igual a '0001'.

Solución:

- Relación:

VENTAS (*cifc*, *dni*, *codcoche*, *color*)

- Variables de dominio:

a(*cifc*), b(*dni*), c(*codcoche*), d(*color*)

- Expresión:

$$\{< b > \mid \neg \exists\, a, c, d\ (< a, b, c, d > \in \text{VENTAS} \wedge \neg\ (a = 0001))\}$$

Este problema se ha planteado como: "obtener los valores del atributo *dni* para los clientes que en la relación VENTAS no les corresponde ningún valor de *cifc* distinto de 0001".

dni
0002

♦ Problema 4.43

Obtener los valores del atributo *codcoche* de aquellos automóviles que han sido comprados por todos los clientes de 'Madrid'.

Solución:

- Relaciones:

CLIENTES (*dni*, *nombre*, *apellido*, *ciudad*)
VENTAS (*cifc*, *dni*, *codcoche*, *color*)

- Variables de dominio:

a(*dni*), b(*CLIENTES.nombre*), c(*apellidos*), d(*CLIENTES.ciudad*),
e(*cifc*), f(*codcoche*), g(*color*)

- Expresión:

$\{< f > \mid \forall a, b, c, d ((< a, b, c, d > \in CLIENTES \wedge d = \text{'Madrid'})$

$\Rightarrow \exists e, g (< e, a, f, g > \in VENTAS))\}$

Como $P_1 \Rightarrow P_2$ equivale a $\neg P_1 \vee P_2$, la expresión anterior se puede poner:

$\{< f > \mid \forall a, b, c, d (\neg (< a, b, c, d > \in CLIENTES \wedge d = \text{'Madrid'})$

$\vee \exists e, g (< e, a, f, g > \in VENTAS))\}$

codcoche

Capítulo 5
SQL

5.1 Introducción

SQL son las iniciales en inglés del lenguaje orientado al manejo de datos denominado *Structured Query Language*. Este lenguaje orientado al sector de las bases de datos, fue desarrollado por IBM para sus máquinas más potentes de la época de los 70. El auge que desde esa fecha ha ido tomando ha sido espectacular y hoy en día, en la mayoría de los gestores de bases de datos se puede encontrar una versión más o menos potente del mismo.

En la actualidad, se puede comparar el SQL al latín medieval es decir, es una lengua universal con la que se puede dialogar con casi todas las máquinas independientemente de los idiomas particulares de cada una de ellas. Esto es debido, como se dijo antes, a que la mayoría de los desarrolladores de sistemas de bases de datos incluyen este lenguaje dentro del producto; ejemplo de algunos de ellos son DB2, ORACLE, INFORMIX, etc.

Hecha esta pequeña introducción, se va a realizar un repaso de los conceptos fundamentales del lenguaje desde el punto de vista sintáctico, de manera que pueda servir como guía de referencia rápida para la resolución de los ejercicios. Tal y como se matizó en el prólogo del libro, es importante que antes de comenzar a resolver los problemas en SQL se haya realizado un esfuerzo por comprender y resolver problemas parecidos utilizando *álgebra relacional*. El motivo de esto es que una vez comprendidas las operaciones a realizar desde el punto de vista matemático, el paso a SQL será una labor prácticamente inmediata ya que se puede considerar al álgebra relacional como el ensamblador del SQL.

5.2 Formas de uso del SQL

La mayoría de los sistemas de bases de datos que incorporan SQL como lenguaje de consulta y manejo de los mismos, ofrecen dos modos posibles de utilizar dicho lenguaje:

- *Interactiva*: Este es un modo mediante el cual se realizan las consultas directamente en la consola o terminal y se obtienen de manera automática las respuestas deseadas (siempre que las preguntas se hayan formulado de manera correcta). Dicho de otro modo, se produce un diálogo en tiempo real entre el sistema y el operador.
- *Inmersa*: Este es el modo de uso habitual cuando se realizan aplicaciones en COBOL, C, etc. Para ello se incluyen los comandos SQL dentro del programa como si se tratase de comandos propios del lenguaje anfitrión. Posteriormente el compilador traducirá los mandatos SQL al lenguaje anfitrión que los absorberá y los ejecutará como parte del propio programa. En este modo los comandos SQL suelen estar precedidos de un carácter especial que le indicará al compilador que se trata de un comando especial.

5.3 Partes del SQL

Los mandatos de SQL pueden dividirse en tres tipos:

- Los que hacen referencia a la definición de datos: creación de bases de datos, diseño de tablas, de vistas, de índices, etc. (DDL *Data Definition Language.*)
- Los que hacen referencia a la manipulación de datos: inserción, modificación y borrado, así como a la consulta de los mismos (DML *Data Manipulation Language*).
- Los que hacen referencia al control y seguridad de los datos: privilegios, controles de acceso, etc.

Siguiendo este orden, se describirán de forma concisa cada uno de todos ellos.

5.4 Tipos de datos

SQL standard maneja los siguientes tipos de datos.

- *Datos numéricos*:

INTEGER	Entero binario de palabra completa
SMALLINT	Entero binario de media palabra
DECIMAL(p,q)	Número decimal de p dígitos y signo con q dígitos a la derecha del punto decimal

FLOAT(p)	Número de punto flotante, con precisión de p dígitos binarios.

- *Datos de cadena*:

CHARACTER(*n*)	Cadena de longitud fija con exactamente n caracteres de 8 bits.
VARCHAR(*n*)	Cadena de longitud variable con hasta *n* caracteres de 8 bits
GRAPHICS(*n*)	Cadena de longitud fija con exactamente *n* caracteres de 16 bits
VARGRAPHICS(*n*)	Cadena de longitud variable con hasta *n* caracteres de 16 bits.

- *Datos de fecha/hora*:

DATE	fecha (aaaammdd)
TIME	hora(hhmmss)
TIMESTAMP	Marca de tiempo. Combinación de fecha y hora con una precisión de microsegundos

En el mundo real, muchas veces surge el problema de la información faltante como por ejemplo, datos que no se conocen, campos vacíos, etc. La forma de representar información faltante en sistemas SQL, es mediante indicadores especiales llamados nulos (NULLS). Si un registro tiene un nulo en una posición de campo, significará que se desconoce el valor del mismo. En SQL cualquier campo puede contener nulos a menos que la definición de ese campo especifique la cláusula NOT NULL de manera explícita.

5.5 Creación de tablas

El comando utilizado en SQL para crear tablas es el siguiente:

```
CREATE TABLE nombre de la tabla
   ( nombre de atributo   tipo de dato,
     nombre de atributo   tipo de dato,...
[PRIMARY  KEY  (clave primaria),]
[FOREING  KEY (clave externa)
REFERENCES (tabla en la que clave externa es clave)] );
```

5.6 Modificación de tablas

También es posible alterar una tabla base ya existente agregando una columna nueva a la derecha mediante la proposición ALTER TABLE que presenta la siguiente sintaxis:

```
ALTER TABLE  nombre de la tabla
ADD nombre del campo  tipo de datos;
```

5.7 Eliminación de tablas

Para eliminar una tabla existente, SQL utiliza el siguiente comando:

```
DROP TABLE  nombre de la tabla;
```

5.8 Índices

Los índices, como las tablas base, se crean y eliminan utilizando proposiciones de definición de datos de SQL. El formato general para crear índices es:

```
CREATE  [UNIQUE]  INDEX  nombre del índice
ON nombre de la tabla  (campo [orden], campo [orden], ...)
[CLUSTER]
```

Es importante resaltar que a la hora de consultar a la base de datos, el usuario no debe hacer referencia a los índices, ya que SQL se encarga automáticamente de decidir cuando se deben utilizar y cuando no.

Para eliminar un índice se utiliza el siguiente comando:

```
DROP  INDEX nombre del índice
```

5.9 Comando SELECT

Entre todos los comandos de SQL probablemente sea SELECT el más utilizado de todos ellos. Su sintaxis completa es la siguiente:

```
SELECT  [DISTINCT]  elemento(s)
FROM  tabla, tabla, ...
[WHERE  condición ]
[GROUP BY  campo(s)]
[HAVING  condición]
[ORDER BY  campo(s)] ;
```

5.10 Filtros en SQL

Filtrar las filas de una o varias tablas, es una operación que se realizará de manera muy frecuente cuando se vaya a extraer información de una base de datos. Esta misión la cumple la cláusula WHERE tal y como se ha mencionado antes utilizando como condición los siguientes símbolos:

< Menor que <= Menor o igual
> Mayor que >= Mayor o igual
!< No menor que !> No Mayor que
= Igual <> ó ¡= ó # Distinto

También se podrán combinar varias condiciones en una misma cláusula WHERE usando los conectores lógicos NOT, AND y OR.

Existen además otros tipos de cláusulas que se colocan detrás de WHERE y que permiten filtrar la información de una manera mucho más sofisticada. Estos predicados son los siguientes:

- BETWEEN: Sirve para comprobar que un valor se encuentra dentro de un rango. Su formato es:

  WHERE [NOT] <expresión>
  [NOT] BETWEEN <valor menor> AND <valor mayor>

- IN: Sirve para comprobar si un valor está dentro de una lista de elementos.

  WHERE campo [NOT] IN ('valores')

  donde 'valores' puede ser también una subconsulta.

- LIKE: Sirve para comparar una columna tipo carácter, con una cadena de caracteres.

  WHERE campo [NOT] LIKE 'cadena'

A la hora de la comparación se dispone de dos símbolos que se pueden utilizar como comodines y que sustituyen a los caracteres que no se conocen o que no interesen. Estos son:

_ Sustituye un carácter de la cadena

% Sustituye algún número de caracteres.

- EXISTS: Es el cuantificador de existencialidad. La expresión

  WHERE EXISTS (SELECT...FROM...)

  resulta verdadera si y solo si el resultado de evaluar la consulta SELECT...FROM... no es el conjunto vacío.

5.11 La operación renombramiento

SQL proporciona un mecanismo para renombrar tanto relaciones como atributos. Para ello se utiliza la cláusula AS que presenta la siguiente sintaxis:

nombre antiguo AS nombre nuevo

La cláusula AS puede aparecer tanto en un SELECT como en un FROM

5.12 Ordenación de resultados

Para ordenar los datos obtenidos en una consulta, se utiliza la cláusula ORDER BY detrás de WHERE.

```
WHERE condición
ORDER BY <columna/integer> [ASC/DESC]
        [,<columna/integer> [ASC/DESC...]
```

5.13 Funciones agregadas

SQL ofrece una serie de funciones de agregados especiales para ampliar su capacidad básica de recuperación de información. Estas funciones son:

COUNT ( ) Cuenta el número de filas devueltas por un comando SQL

SUM ( )	Devuelve la suma de la columna o expresiones especificadas
AVG ( )	Sirve para calcular la media de la columna o expresión especificada
MAX ( )	Devuelve el valor más alto de la columna o expresión especificada.
MIN ( )	Devuelve el valor más bajo de la columna o expresión especificada

5.14 Agrupación de filas

El operador GROUP BY reorganiza en sentido lógico la tabla representada por la cláusula FROM formando grupos de manera que dentro de un grupo dado, todas las filas tengan el mismo valor en el campo de GROUP BY.

```
SELECT <columna/expresión>
FROM tablas
WHERE <condición>
GROUP BY <columna/expresión> [HAVING <condición>];
```

Cuando se usa la cláusula GROUP BY, los nombres de columnas que aparecen en SELECT deben especificarse también en esta cláusula.

La cláusula HAVING funciona en cada grupo como WHERE en cada fila.

5.15 Problemas resueltos

En esta sección se proponen y resuelven problemas utilizando SQL. Mientras no se diga lo contrario, en los problemas se hace referencia a la base de datos AUTOMÓVILES descrita en el apéndice A. En la solución de los problemas, los nombres de las tablas y campos aparecen siempre en minúscula para distinguirlos de los comandos SQL propiamente dichos que se muestran en mayúsculas.

♦ Problema 5.1

Crear la tabla COCHES.

Solución:

```
CREATE TABLE coches
      (codcoche        SMALLINT        NOT NULL,
       nombre          CHAR(10)        NOT NULL,
       modelo          CHAR(10)        NOT NULL,
PRIMARY KEY  (codcoche));
```

♦ Problema 5.2

Crear la tabla DISTRIBUCIÓN.

Solución:

```
CREATE TABLE distribución
      (cifc            SMALLINT        NOT NULL,
       codcoche        SMALLINT        NOT NULL,
       cantidad        SMALLINT        NOT NULL,
PRIMARY KEY (cifc, codcoche),
FOREING KEY (cifc) REFERENCES (concesionario),
FOREING KEY (codcoche) REFERENCES (coches));
```

♦ Problema 5.3

Crear un índice en la tabla MARCAS llamado *icifm* ,del campo *cifm*.

Solución:

```
CREATE UNIQUE INDEX icifm ON marcas (cifm)
```

♦ Problema 5.4

Crear un índice en la tabla COCHES llamado *itodos*, de todos los campos.

Solución:

```
CREATE UNIQUE INDEX itodos ON coches (codcoche, nombre, modelo)
```

♦ Problema 5.5

Añadir en la tabla MARCAS un nuevo campo llamado *pais*.

Solución:

```
ALTER TABLE marcas ADD pais CHAR(15)
```

♦ Problema 5.6

Borrar la tabla DISTRIBUCIÓN.

Solución:

```
DROP TABLE distribucion
```

♦ Problema 5.7

Obtener todos los valores de todos los concesionarios.

Solución:

```
SELECT *
FROM concesionario;
```

cifc	*nombre*	*ciudad*
0001	*acar*	*Madrid*
0002	*bcar*	*Madrid*
0003	*ccar*	*Barcelona*
0004	*dcar*	*Valencia*
0005	*ecar*	*Bilbao*

♦ Problema 5.8

Obtener todos los campos de todos los clientes de 'Madrid'.

Solución:

```
SELECT *
FROM   clientes
WHERE ciudad='Madrid';
```

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	Garcia	Madrid
0003	Juan	Martin	Madrid
0004	Maria	Garcia	Madrid

♦ Problema 5.9

Obtener los nombres de todas las marcas de coches ordenadas alfabéticamente.

Solución:

```
SELECT    nombre
FROM      marcas
ORDER BY  nombre;
```

Nombre
audi
bmw
citroen
opel
renault
seat

♦ Problema 5.10

Obtener el *cifc* de todos los concesionarios cuyo atributo *cantidad* en la tabla de DISTRIBUCION es mayor que 18.

Solución:

```
SELECT cifc
FROM   distribucion
WHERE  cantidad > 18;
```

cifc
0005

♦ Problema 5.11

Obtener el *cifc* de todos los concesionarios cuyo atributo *cantidad* en la tabla de DISTRIBUCIÓN, está comprendido entre 10 y 18 ambos inclusive.

Solución:

```
SELECT cifc
FROM   distribucion
WHERE  cantidad BETWEEN 10 AND 18;
```

cifc
0002
0002
0004
0005
0005

♦ Problema 5.12

Obtener el *cifc* de todos los concesionarios cuyo atributo *cantidad* en la tabla de DISTRIBUCIÓN, está comprendida entre 10 y 18, ambos inclusive. (De otra manera).

Solución:

```
SELECT cifc
FROM   distribucion
WHERE  cantidad >= 10
AND    cantidad <= 18;
```

♦ Problema 5.13

Obtener el *cifc* de todos los concesionarios que han adquirido más de 10 coches o menos de 5 de algún tipo (no en total).

Solución:

```
SELECT cifc
FROM   distribucion
WHERE  cantidad >10
OR     cantidad <5;
```

cifc
0001
0003
0005
0006

♦ Problema 5.14

Obtener todas las parejas de *cifm* de marcas y *dni* de clientes que sean de la misma ciudad.

Solución:

Para resolver este problema es necesario reunir las tablas MARCAS y CLIENTES según el campo *ciudad* y después proyectar sobre los campos pedidos.

```
SELECT cifm, dni
FROM   marcas, clientes
WHERE  marcas.ciudad = clientes.ciudad;
```

cifm	*dni*
0001	0001
0001	0003
0001	0004
0002	0005

(continuación)

cifm	*dni*
0002	0006
0003	0002
0004	0001

(continuación)

cifm	*dni*
0004	0003
0004	0004
0006	0005

(continuación)

cifm	*dni*
0006	0006
0005	0003
0006	0004

♦ Problema 5.15

Obtener todas las parejas de *dni* de clientes y *cifm* de marcas que NO sean de la misma ciudad.

Solución:

```
SELECT    cifm, dni
FROM      marcas, clientes
WHERE NOT (marcas.ciudad = clientes.ciudad);
```

cifm	*dni*
0001	0002
0001	0005
0001	0006
0002	0001
0002	0002
0002	0003
0002	0004
0003	0001
0003	0003
0003	0004
0003	0005
0003	0006

cifm	*dni*
0004	0002
0004	0005
0004	0006
0005	0001
0005	0002
0005	0003
0005	0004
0005	0005
0005	0006
0006	0001
0006	0002
0006	0003
0006	0004

♦ Problema 5.16

Obtener los *codcoche* suministrados por algún concesionario de 'Barcelona'

Solución:

```
SELECT codcoche
FROM   distribucion, concesionarios
WHERE  distribucion.cifc=concesionarios.cifc
AND    concesionarios.ciudad='Barcelona';
```

codcoche
010
011
012

♦ Problema 5.17

Obtener el *codcoche* de aquellos coches vendidos a clientes de 'Madrid'.

Solución:

Para resolver este problema, habrá primero que calcular el *dni* de los clientes que son de Madrid y después ver cuáles de ellos se encuentran en la tabla VENTAS. Esto se puede hacer de varias maneras aunque en este ejercicio lo haremos como una simple reunión entre tablas, es decir;

```
SELECT codcoche
FROM   ventas, clientes
WHERE  clientes.ciudad='Madrid'
AND    clientes.dni=ventas.dni;
```

codcoche
001
005
011
014

♦ Problema 5.18

Obtener el *codcoche* de los coches que han sido adquiridos por un cliente de 'Madrid' en un concesionario de 'Madrid'.

Solución:

Para resolver este problema, será necesario utilizar tres tablas, VENTAS, CONCESIONARIOS y CLIENTES. El proceso de resolución será el siguiente: encontrar los clientes de Madrid, encontrar los concesionarios de Madrid y obtener los *codcoche* que en la tabla de VENTAS tienen a su lado un *cifc* de los hallados anteriormente con un *dni* de los clientes que residen en Madrid. Esto en SQL se puede realizar de diversas maneras aunque en este ejercicio lo haremos como una mera reunión entre tablas.

```
SELECT codcoche
FROM   ventas, clientes, concesionarios
WHERE  concesionarios.ciudad='Madrid'
AND    clientes.ciudad='Madrid'
AND    concesionarios.cifc=ventas.cifc
AND    clientes.dni=ventas.dni;
```

Básicamente lo que se ha hecho es realizar la reunión natural de las tablas CONCESIONARIOS, VENTAS y CLIENTES, según los campos comunes *cifc* y *dni* y luego se han seleccionado aquellas tuplas cuyos campos *concesionario.ciudad* y *clientes.ciudad* tuviesen un valor igual a 'Madrid'.

Codcoche
001
008
006

♦ Problema 5.19

Obtener los *codcoche* de los coches comprados en un concesionario de la misma ciudad que el cliente que lo compra.

Solución:

La manera de realizar este problema es similar al anterior, aunque ahora lo que hay que hacer es imponer que las ciudades del concesionario y del cliente sean las mismas.

```
SELECT codcoche
FROM   ventas, clientes, concesionarios
WHERE  concesionarios.ciudad=clientes.ciudad
AND    concesionarios.cifc=ventas.cifc
AND    clientes.dni=ventas.dni;
```

codcoche
001
008
006

♦ Problema 5.20

Obtener los *codcoche* de los coches comprados en un concesionario de distinta ciudad que el cliente que lo compra

Solución:

```
SELECT  codcoche
FROM    ventas, clientes, concesionario

WHERE   concesionarios.ciudad<>clientes.ciudad
AND     concesionarios.cifc=ventas.cifc
AND     clientes.dni=ventas.dni;
```

codcoche
005
011
014

♦ Problema 5.21

Obtener todas las parejas de nombre de marcas que sean de la misma ciudad.

Solución:

El proceso de resolución de este problema consistirá en realizar una reunión natural de la tabla MARCAS consigo misma según el campo *ciudad*, para después seleccionar las tuplas resultantes y proyectar sobre los campos *nombre*. La dificultad reside en que en SQL una tabla no se puede reunir consigo misma (p.e. marcas.ciudad = marcas.ciudad) ya que esta condición siempre es verdadera. Para ello lo que hay que hacer es darle un alias a la tabla de manera que es como si se trabajara con dos tablas distintas.

En este caso, a la tabla MARCAS se le darán dos alias distintos (a y b) de manera que a efectos formales es como si se tratara de dos tablas totalmente distintas.

```
SELECT a.nombre, b.nombre
FROM   marcas a, marcas b
WHERE  a.ciudad = b.ciudad
AND    a.nombre < b.nombre;
```

a.nombre	b.nombre
audi	seat
bmw	renault

La condición a.nombre < b.nombre se pone para que no se repitan las parejas como por ejemplo (audi, seat) y (seat, audi).

♦ Problema 5.22

Obtener las parejas de modelos de coches cuyo nombre es el mismo y cuya marca es de 'Bilbao'.

Solución:

La manera de solucionar este problema es muy parecida al anterior, aunque con la dificultad añadida de que hay que manejar varias tablas. Lo primero, al igual que antes, será reunir la tabla COCHES consigo misma según el campo *nombre*. Una vez hecho esto, habrá que reunir este resultado con la tabla MARCO para ver cuál es la marca de cada coche, y el resultado global reunirlo con la tabla MARCAS para ver cual es la ciudad de cada marca. Una vez realizado esto sólo quedará seleccionar aquellas tuplas cuya marcas.ciudad sea igual a BILBAO y proyectar sobre los campos *modelo*.

```
SELECT a.modelo, b.modelo
FROM   coches a, coches b, marco, marcas
WHERE  a.nombre = b.nombre
AND    a.modelo > b.modelo
AND    a.codcoche = marco.codcoche
AND    marco.cifm = marcas.cifm
AND    marcas.ciudad= 'Bilbao';
```

a.modelo	*b.modelo*
caravan	gti

♦ Problema 5.23

Obtener todos los *codcoche* de los coches cuyo nombre empiece por '*c*'.

Solución:

```
SELECT codcoche
FROM   coches
WHERE  nombre LIKE 'c%';
```

codcoche
005
017

♦ Problema 5.24

Obtener todos los codcoche de los coches cuyo nombre no contiene ninguna '*a*'.

Solución:

```
SELECT DISTINCT codcoche
FROM   coches
WHERE  nombre NOT LIKE '%a%';
```

codcoche
004
010
011
018
019
020

♦ Problema 5.25

Obtener el número total de nombres de marcas de coches que son de Madrid.

Solución:

```
SELECT COUNT(DISTINCT nombre)
FROM   marcas
WHERE  ciudad = 'Madrid';
```

COUNT (nombre)
2

♦ Problema 5.26

Obtener la media de la cantidad de coches que tienen todos los concesionarios.

Solución:

```
SELECT AVG (cantidad)
FROM   distribucion;
```

AVG (cantidad)
7.7

♦ Problema 5.27

Obtener el *dni* cuya numeración sea la más alta de todos los clientes de 'Madrid'.

Solución:

```
SELECT MAX(dni)
FROM   clientes
WHERE  ciudad='Madrid';
```

MAX (dni)
0004

♦ Problema 5.28

Obtener el *dni* con numeración mas baja de todos los clientes que han comprado un coche 'blanco'.

Solución:

```
SELECT MIN(dni)
FROM   ventas
WHERE  color='blanco'
```

MIN (dni)
0001

♦ Problema 5.29

Obtener el *cifc* de todos los concesionarios cuyo número de coches en stock no es nulo.

Solución

```
SELECT DISTINCT cifc
FROM   distribucion
WHERE  cantidad  IS NOT NULL;
```

cifc
0001
0002
0003

(continuación)

cifc
0004
0005

♦ Problema 5.30

Obtener el *cifm* y el *nombre* de las marcas de coches cuya segunda letra del nombre de la ciudad de origen sea una 'i'.

Solución:

```
SELECT DISTINCT   cifm, nombre
FROM              marcas
WHERE             ciudad LIKE '_i%';
```

cifm	*nombre*
0005	opel

♦ Problema 5.31

Obtener el *dni* de los clientes que han comprado algún coche a un concesionario de 'Madrid'.

Solución:

```
SELECT DISTINCT dni
FROM   ventas
WHERE  cifc IN  (SELECT  cifc
                 FROM    concesionarios
                 WHERE   ciudad = 'Madrid');
```

dni
0001
0002
0003

♦ Problema 5.32

Obtener el color de los coches vendidos por el concesionario 'acar'.

Solución:

Este, al igual que el problema anterior, puede resolverse tal y como se hizo en los problemas 5.14-5.22 es decir, realizando reuniones naturales entre las tablas.

```
SELECT DISTINCT color
FROM   ventas, concesionarios
WHERE  concesionarios.nombre = 'acar'
AND    concesionarios.cifc = ventas.cifc;
```

No obstante éste y los siguientes problemas, se resolverán utilizando subconsultas.

```
SELECT DISTINCT color
FROM   ventas
WHERE cifc IN (SELECT cifc
               FROM   concesionarios
               WHERE  nombre = 'acar');
```

color
blanco
rojo

♦ Problema 5.33

Obtener el *codcoche* de los coches vendidos por algún concesionario de 'Madrid'.

Solución:

```
SELECT DISTINCT codcoche
FROM   ventas
WHERE cifc IN  (SELECT  cifc
                FROM    concesionarios
                WHERE   ciudad = 'Madrid');
```

codcoche
001
005
008
006

♦ Problema 5.34

Obtener el nombre y el modelo de los coches vendidos por algún concesionario de 'Barcelona'.

Solución:

```
SELECT nombre, modelo
FROM   coches
WHERE codcoche IN  (SELECT codcoche
                    FROM ventas
                    WHERE cifc IN (SELECT cifc
                                   FROM concesionarios
                                   WHERE ciudad = 'Barcelona'));
```

nombre	*modelo*
ZX	TD

♦ Problema 5.35

Obtener todos los nombres de los clientes que hayan adquirido algún coche del concesionario 'dcar'.

Solución:

```
SELECT nombre
FROM   clientes
WHERE  dni IN (SELECT dni
               FROM   ventas
               WHERE  cifc IN (SELECT cifc
                               FROM concesionarios
                               WHERE nombre = 'dcar'));
```

nombre
Javier

♦ Problema 5.36

Obtener el *nombre* y el *apellido* de los clientes que han adquirido un coche modelo 'gti' de color 'blanco'.

Solución:

```
SELECT nombre, apellido
FROM   clientes
WHERE dni IN (SELECT dni
              FROM ventas
              WHERE color = 'blanco'
AND  codcoche IN (SELECT codcoche
               FROM coches
               WHERE  modelo = 'gti'));
```

nombre	*apellido*

Es decir, no existe ninguno.

♦ Problema 5.37

Obtener el nombre y el apellido de los clientes que han adquirido un automóvil a un concesionario que posea actualmente coches en stock del modelo 'gti'.

Solución:

```
SELECT nombre, apellido
FROM   clientes
WHERE  dni  IN
            (SELECT dni
             FROM   ventas
             WHERE  cifc  IN
                        (SELECT cifc
                        FROM   distribucion
                        WHERE  codcoche IN
                                    (SELECT codcoche
                                     FROM   coches
                                     WHERE  modelo = 'gti')));
```

nombre	*apellido*
Luis	Garcia
Antonio	Lopez

♦ Problema 5.38

Obtener el nombre y el apellido de los clientes que han adquirido un automóvil a un concesionario de 'Madrid' que posea actualmente coches en stock del modelo 'gti'.

Solución:

```
SELECT nombre, apellido
FROM   clientes
WHERE  dni  IN
          (SELECT  dni
          FROM  ventas
          WHERE cifc IN
                    (SELECT cifc
                    FROM    concesionarios
                     WHERE  ciudad = 'Madrid'
                     AND cifc IN
                           (SELECT  cifc
                           FROM    distribucion
                           WHERE  codcoche IN
```

```
                                    (SELECT codcoche
                                     FROM   coches
                                     WHERE modelo = 'gti'))));
```

nombre	*Apellido*
Luis	García

♦ Problema 5.39

Obtener el nombre y el apellido de los clientes cuyo número de dni es menor que el del cliente 'Juan Martín'.

Solución:

```
SELECT nombre, apellido
FROM   clientes
WHERE  dni < (SELECT  dni
              FROM  clientes
              WHERE nombre = 'Juan'
              AND apellido = 'Martin');
```

nombre	*apellido*
Luis	Garcia
Antonio	Lopez

♦ Problema 5.40

Obtener el nombre y el apellido de los clientes cuyo dni es menor que el de los clientes que son de 'Barcelona'.

Solución:

```
SELECT nombre, apellido
FROM   clientes
WHERE  dni < ALL (SELECT  dni
                  FROM  clientes
                  WHERE ciudad = 'Barcelona');
```

nombre	*apellido*
Luis	Garcia
Antonio	Lopez
Juan	Martin
Maria	Garcia

♦ Problema 5.41

Obtener el nombre y el apellido de los clientes cuyo nombre empieza por 'a' y cuyo número del dni es mayor que el de todos los clientes que son de 'Madrid'.

Solución:

```
SELECT nombre, apellido
FROM   clientes
WHERE  dni > ALL (SELECT  dni
                  FROM    clientes
                  WHERE   ciudad = 'Madrid')
AND  nombre LIKE  'a%';
```

nombre	*apellido*
Ana	Lopez

♦ Problema 5.42

Obtener el nombre y el apellido de los clientes cuyo nombre empieza por 'a' y cuyo número del dni es mayor que el de *alguno* de los clientes que son de 'Madrid'.

Solución:

```
SELECT nombre, apellido
FROM   clientes
WHERE  dni > ANY (SELECT  dni
                  FROM    clientes
                  WHERE   ciudad = 'Madrid')
AND      nombre LIKE  'a%';
```

nombre	*apellido*
Antonio	Lopez
Ana	Lopez

♦ Problema 5.43

Obtener el nombre y el apellido de los clientes cuyo nombre empieza por 'A' y cuyo dni es mayor que el de *alguno* de los clientes que son de 'Madrid' o menor que el de todos los de 'Valencia'.

Solución:

```
SELECT nombre, apellido
FROM   clientes
WHERE  (dni > ANY (SELECT dni
                   FROM   clientes
                   WHERE  ciudad = 'Madrid')
        OR dni < ALL (SELECT dni
                      FROM clientes
                      WHERE  ciudad = 'Valencia'))
AND nombre LIKE 'A%';
```

¡ Cuidado con los paréntesis !

nombre	*apellido*
Antonio	Lopez
Ana	Lopez

♦ Problema 5.44

Obtener el nombre y el apellido de los clientes que han comprado como mínimo un coche 'blanco' y un coche 'rojo'.

Solución:

Este problema, aunque a priori parece trivial, presenta una dificultad adicional; supóngase que se realiza la siguiente consulta:

```
SELECT nombre, apellido
FROM   clientes
WHERE dni IN (SELECT  dni
              FROM    ventas
              WHERE   color = 'blanco')

AND  color = 'rojo');
```

El resultado de la misma sería el conjunto vacío ya que la variable *color* en cada tupla no puede valer a la vez 'blanco' y 'rojo'. Una de las posibles soluciones es la siguiente:

```
SELECT nombre, apellido
FROM   clientes
WHERE dni IN (SELECT  dni
              FROM ventas
```

```
                WHERE color = 'blanco')
AND dni IN (SELECT dni
            FROM ventas
            WHERE color = 'rojo');
```

Esta sería una de las maneras correctas de realizar la consulta ya que así el operador SELECT seleccionará aquellos nombres y apellidos de aquellas tuplas cuyo atributo *dni* tenga un valor que esté contenido en los dos grupos seleccionados por los otros dos SELECT.

nombre	*apellido*
Luis	García

♦ Problema 5.45

Obtener el dni de los clientes cuya ciudad sea la última de la lista alfabética de las ciudades donde hay concesionarios.

Solución:

```
SELECT dni
FROM   clientes
WHERE  ciudad = (SELECT  MAX(ciudad)
                 FROM  concesionario);
```

dni
0002

♦ Problema 5.46

Obtener la media de los automóviles que cada concesionario tiene actualmente en stock.

Solución:

Para resolver este tipo de problemas hay que pensar en cómo se haría a mano. Para ello, sería necesario dividir la tabla en grupos, estando compuestos cada uno de ellos por tuplas con el mismo valor del atributo *cifc*. Ahora en cada uno

de esos grupos se calcularía la media aritmética de su atributo *cantidad*. Todo eso es posible hacerlo en SQL utilizando las funciones agregadas y GROUP BY.

```
SELECT    cifc, AVG(cantidad)
FROM      distribucion
GROUP BY cifc;
```

cifc	*AVG(cantidad)*
0001	5,7
0002	8,3
0003	4,3
0004	7,5
0005	12,7

♦ Problema 5.47

Obtener el *cifc* del concesionario que no sea de 'Madrid' cuya media de vehículos en stock sea la mas alta de todas las medias.

Solución:

```
SELECT cifc
FROM   concesionarios
WHERE  ciudad <> 'Madrid'
AND    cifc IN (SELECT cifc
                FROM   distribucion
                GROUP  BY cifc
                HAVING AVG (cantidad) >= (SELECT AVG(cantidad)
                                          FROM distribucion
                                          GROUP BY cifc));
```

cifc
0005

♦ Problema 5.48

Repetir el ejercicio 5.33 pero utilizando el comando EXISTS en la solución.

Solución:

```
SELECT DISTINCT  codcoche
FROM ventas
WHERE  EXISTS (SELECT *
```

```
FROM concesionario
WHERE ciudad = 'Madrid'
AND concesionario.cifc = ventas.cifc);
```

codcoche
0001
0005
0008
0006

Como puede verse, en la solución anterior la formulación del problema es muy similar a la del problema 5.33, aunque conceptualmente es muy distinta ya que la manera de operar es la siguiente:

Para cada tupla de la tabla *ventas*, se seleccionará su atributo *codcoche* si existe en la tabla *concesionario* alguna tupla cuyo campo *ciudad* tenga como valor MADRID y cuyo campo *cifc* sea igual al de la tupla de la tabla *ventas* en cuestión.

♦ Problema 5.49

Utilizando EXISTS, obtener el dni de los clientes que hayan adquirido por lo menos alguno de los coches que ha sido vendido por el concesionario cuyo cifc es 0001.

Solución:

```
SELECT DISTINCT va.dni
FROM   ventas  va
WHERE EXISTS (SELECT *
                FROM ventas  vb
                WHERE va.codcoche = vb.codcoche
                AND vb.cifc = 0001);
```

Como antes, la explicación de este problema es la siguiente: para cada tupla de la tabla VA (que es un alias de VENTAS) se seleccionará el atributo *dni*, si existe en la tabla VB (que es otro alias de VENTAS), alguna tupla cuyo atributo *cifc* tenga como valor 0001 y cuyo atributo *codcoche* sea el mismo que la tupla de la tabla VA en cuestión. El motivo de crear los alias es que para resolver este problema es necesario realizar una reunión de la tabla

VENTAS consigo misma (va.codcoche = vb.codcoche), de manera que si no fuese por los alias esta condición siempre sería cierta.

dni
0001
0002

♦ Problema 5.50

Obtener los dni de los clientes que sólo han comprado coches al concesionario 0001.

Solución:

Este tipo de problemas ("Obtener X de los que sólo...a Y) es mejor intentar reformularlos al revés para que la solución sea más sencilla, es decir, en base a negaciones en vez de afirmaciones. En este caso concreto el problema se podría plantear como: *"Obtener los dni de los clientes que en la tabla VENTAS no tienen a su lado un valor de cifc DISTINTO de* 0001*"*.

Esta frase, es posible expresarla de una manera más cercana al SQL como:

"Seleccionar los dni de las tuplas de los clientes de la tabla VENTAS, de manera que no exista para esa tupla en el atributo cifc un valor distinto del 0001*"*.

Puesto esto en formato SQL la pregunta quedaría:

```
SELECT dni
FROM   ventas va
WHERE  NOT EXISTS (SELECT *
                   FROM   ventas vb
                   WHERE  cifc <> 0001
                   AND    va.dni = vb.dni);
```

dni
0002

♦ Problema 5.51

Obtener los nombres de los clientes que no han comprado ningún coche 'rojo' a ningún concesionario de 'Madrid'.

Solucion:

Este es otro problema que se resuelve de manera muy sencilla utilizando el comando NOT EXISTS. Para ello el problema se puede plantear como

"Obtener el nombre de los clientes cuyo dni no aparezca en la tabla VENTAS al lado de un codcoche cuyo color sea rojo y al lado de un cifc que sea de Madrid".

```
SELECT  nombre
FROM clientes
WHERE  NOT EXISTS (SELECT *
                   FROM ventas
                   WHERE ventas.dni = clientes.dni
                   AND color = 'rojo'
                   AND  cifc IN (SELECT cifc
                                 FROM  concesionarios
                                 WHERE ciudad = 'Madrid'));
```

nombre
Juan
Maria
Javier
Ana

♦ Problema 5.52

Obtener el nombre de los clientes que sólo han comprado en el concesionario de cifc 0001.

Solución:

Este problema, al igual que los anteriores, es mucho mejor formularlo de manera inversa, es decir:

"Obtener el nombre de los clientes cuyo dni (que está en la tabla VENTAS) no aparece en la tabla VENTAS al lado de algún cifc distinto de 0001*"*

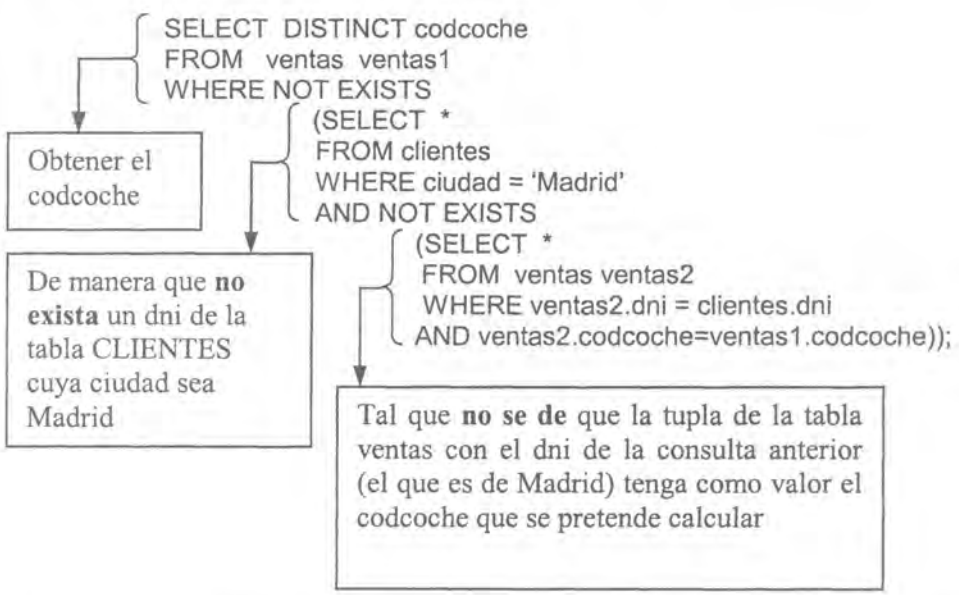
```
SELECT nombre
FROM  clientes
WHERE dni IN (SELECT dni
              FROM  ventas  ventas1
              WHERE NOT EXISTS (SELECT *
                                FROM  ventas  ventas2
                                WHERE ventas1.dni=ventas2.dni
                                AND ventas2.cifc<>0001));
```

nombre
Antonio

♦ Problema 5.53

Obtener el codcoche de aquellos automóviles que han sido comprados por todos los clientes de 'Madrid'.

Solución:



Es decir, todos los clientes que son de Madrid deben haber comprado estos coches. En este problema al igual que en los anteriores, es mucho más factible plantearse la pregunta al revés, es decir:

"Obtener los codcoche de los automóviles de manera que no exista un cliente que sea de Madrid que no esté en la tabla VENTAS a su lado (al lado del codcoche)".

Como puede deducirse del planteamiento anterior, el problema es fácilmente abordable utilizando dos negaciones en forma de NOT EXISTS.

codcoche

♦ Problema 5.54

Obtener el codcoche de aquellos automóviles de color 'rojo' y de modelo 'gti' que han sido comprados por todos los clientes cuyo apellido comienza por 'g'.

Solución:

Este es un problema exactamente igual al anterior, pero con algún filtro más en cuanto a las condiciones que debe cumplir el coche y el cliente. Por tanto, el problema planteado al revés sería:

"Obtener los codcoche de los automóviles rojos y gti's de manera que no exista un cliente cuyo apellido comienza por "g" que no esté en la tabla VENTAS a su lado (al lado del codcoche)"

```
SELECT  DISTINCT codcoche
FROM   ventas  ventas1
WHERE codcoche IN
                    (SELECT codcoche
                    FROM  coches
                    WHERE  modelo = 'gti')
AND  color = 'rojo'
AND NOT EXISTS
                (SELECT  *
                FROM clientes
                WHERE apellido LIKE 'g%'
                AND NOT EXISTS
                                (SELECT  *
                                 FROM  ventas ventas2
                                 WHERE ventas2.dni = clientes.dni
                                 AND ventas2.codcoche=ventas1.codcoche));
```

codcoche

♦ Problema 5.55

Obtener el dni de los clientes que han adquirido por lo menos los mismos automóviles que el cliente "Luis García".

Solución:

Este problema es del mismo tipo que los anteriores, por ello para su resolución es mucho mejor formularlo al revés, es decir mediante negaciones:

"Obtener el dni de los clientes de manera que en la tabla VENTAS no exista el codcoche de un automóvil adquirido por Luis Garcia que no aparezca en una tupla de la tabla ventas con este dni".

```
SELECT DISTINCT dni
FROM ventas ventas1
WHERE NOT EXISTS
            (SELECT codcoche
             FROM ventas ventas2
           WHERE dni IN
                        (SELECT dni
                        FROM clientes
                        WHERE nombre = 'Luis'
                        AND apellido = 'Garcia')
       AND NOT EXISTS
                        (SELECT *
                        FROM ventas ventas3
                        WHERE ventas3.codcoche = ventas2.codcoche
                        AND ventas3.dni = ventas1.dni));
```

dni

♦ Problema 5.56

Obtener el cifc de los concesionarios que han vendido el mismo coche a todos los clientes.

Solución:

Este probablemente sea el ejercicio en SQL más complicado de los planteados hasta el momento. Al igual que en los problemas anteriores, es recomendable intentar resolverlo formulando las preguntas mediante negaciones. En este caso este problema podría plantearse de la siguiente manera:

"Obtener el cifc del concesionario de manera que para algún codcoche de la tabla ventas no exista ningún dni de la tabla clientes, que no estén los tres en la misma tupla en la tabla VENTAS".

```
SELECT  DISTINCT  cifc
FROM ventas V1
WHERE EXISTS
        (SELECT codcoche
        FROM  ventas  V2
        WHERE NOT EXISTS
                (SELECT  dni
                FROM clientes
                WHERE  NOT EXISTS
                        (SELECT *
                        FROM ventas V3
                        WHERE V3.cifc=V1.cifc
                        AND V3.codcoche=V2.codcoche
                        AND V3.dni=clientes.dni)));
```

cifc

- **Problema 5.57**

Convertir la siguiente proposición de SQL a su equivalente en Castellano.

```
SELECT DISTINCT dni
FROM ventas  V1
WHERE NOT EXISTS
        (SELECT *
        FROM ventas  V2
        WHERE ventas2.dni=ventas1.dni
        AND NOT EXISTS
                (SELECT *
                FROM ventas V3
                WHERE V3.codcoche=V2.codcoche
                AND V3.cifc=0001));
```

Solución:

"Obtener los dni de los clientes que hayan comprado sólo automóviles al concesionario de cifc 0001".

Capítulo 6
Query-By-Example

6.1. Introducción

Query-by-Example (QBE, consulta mediante ejemplo) es un lenguaje relacional de manipulación de datos, desarrollado por IBM, que forma parte, como una de las interfaces de usuario, del producto de consultas *Query Management Facility* (QMF). Existe una correspondencia muy cercana entre QBE y el *cálculo relacional de dominios*.

En el presente capítulo se proponen y resuelven problemas de QBE y se realiza un recordatorio de los aspectos teóricos más importantes en relación con los problemas.

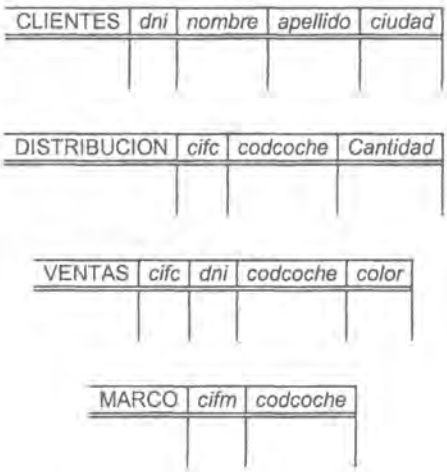
6.2. Consultas en Query-by-Example

En QBE las consultas se expresan a partir de *tablas esqueleto* que muestran el esquema de las relaciones que componen la base de datos. Así, para la base de datos descrita en el apéndice A, sus tablas esqueleto son las siguientes:

MARCAS	*cifm*	*nombre*	*ciudad*

COCHES	*codcoche*	*nombre*	*modelo*

CONCESIONARIOS	*cifc*	*nombre*	*ciudad*



Para realizar una consulta en QBE, primero se seleccionan las tablas esqueleto de las relaciones que se precisan para la consulta y a continuación se rellenan dichas tablas con *filas ejemplo*.

Una *fila ejemplo* está formada por constantes y variables de dominio. Las variables de dominio van precedidas de subrayado *_x* y las constantes aparecen solas sin entrecomillar *c*. En una fila ejemplo también pueden aparecer los operadores que se indican en este capítulo.

Cuando una posición de un campo de una tabla esqueleto aparece en blanco, ello quiere decir que la variable de dominio de dicho campo es única, es decir, sólo aparece una vez en la consulta.

En ocasiones no es conveniente o resulta imposible expresar todos los límites de las variables de dominio dentro de las tablas esqueleto. Así, cuando es necesario comparar dos variables con distintos nombres, dicha comparación no se puede realizar en las tablas esqueleto. Para superar estas limitaciones, QBE introduce el *cuadro de condiciones*.

Condiciones
condición 1
condición 2
...
condición n

El *cuadro de condiciones* es una tabla en cuyas filas aparecen las condiciones necesarias para realizar una determinada consulta.

6.3. Operaciones permitidas en Query-by-Example

En las tablas esqueleto y/o en el cuadro de condiciones, pueden aparecer los operadores que se describen en esta sección.

- *Operador P.* El operador P. se utiliza en las tablas esqueleto para presentar valores de los atributos que aparecen en ellas. Así, cuando en la relación resultante de una consulta deban aparecer los valores de un determinado atributo, se escribe P. en la columna correspondiente de la tabla esqueleto de la relación que contiene dicho atributo. Por otra parte, cuando se quieran presentar todos los campos de una relación, en lugar de poner la orden P. en todas las columnas de la correspondiente tabla esqueleto, se puede poner dicha orden en la columna correspondiente al nombre de la relación. Si en una fila de un atributo que se quiere presentar aparecen varios operadores, el operador P. se situará en primer lugar.

- *Operador ALL.* QBE tiene la particularidad de realizar la eliminación de duplicados automáticamente. Para suprimir en una presentación la eliminación de duplicados, se debe insertar la orden ALL. después de la orden P.

- *Operadores de comparación.* Cuando se realiza una comparación en las tablas esqueleto, el espacio de la parte izquierda de la comparación debe estar en blanco, lo que implica que para poder comparar dos variables éstas deben tener el mismo nombre. Sin embargo, en el cuadro de condiciones si es posible realizar comparaciones entre variables de dominio con nombre distintos. Las comparaciones permitidas en QBE son:

Operador	*Símbolo*
menor que	$<$
menor o igual que	$\leq$
igual que	$=$
mayor que	$>$
mayor o igual que	$\geq$
distinto que	$\neg$ o $\neg =$

- *Operadores aritméticos.* En QBE se pueden utilizar expresiones aritméticas. En dichas expresiones, pueden aparecer distintos operadores aritméticos junto con variables y constantes. Cuando en una comparación aparece una expresión aritmética, ésta debe estar en la parte derecha de la operación de comparación. En los problemas se considerarán los siguientes operadores aritméticos:

Operador	Símbolo
suma	+
resta	-
producto	*

- *Operadores lógicos*. Los operadores lógicos permitidos en QBE son:

Operador	Símbolo
"y" lógico	and o &
"o" lógico	or o |
Negación	not o ¬

Las operaciones lógicas *and* y *or* sólo pueden aparecer en el cuadro de condiciones. Por otra parte, el signo de negación ¬ puede tener distintos significados según sea su situación. Cuando aparece debajo del nombre de una relación y al lado de una fila ejemplo, puede leerse como *no existe*. Mientras que al aparecer debajo del nombre de un atributo, es equivalente a *no igual*.

- *Operadores de agregados y operador UNQ*. QBE incluye los operadores de agregados siguientes:

Operador	Símbolo
media aritmética	AVG.
máximo	MAX.
mínimo	MIN.
suma	SUM.
cuenta	CNT.

Estos operadores se deben poner a la izquierda de ALL. para asegurar que todos los valores apropiados son considerados (no se eliminan duplicados). Cuando sea necesario eliminar los valores duplicados, se utiliza el operador UNQ. situado entre el operador de agregados y el operador ALL.

- *Operador G*. Este operador permite el cálculo de funciones en grupos de tuplas. El operador G., que es análogo a *group by* en SQL, permite reorganizar la relación en que aparece formando grupos de manera que dentro de un grupo dado todas las filas tengan el mismo valor en el campo donde aparece.

- *Operadores AO y DO*. QBE permite controlar el orden en que se presentan las tuplas en una relación. El operador AO. se utiliza para ordenar las tuplas de forma ascendente y el operador DO. para ordenarlas de forma

descendente. La ordenación se puede hacer por múltiples columnas añadiendo un entero entre paréntesis que indica el número de orden de las columnas.

- *Operadores para la modificación de la base de datos.* Estos operadores permiten suprimir D., insertar I. y actualizar U. la información de la base de datos.

  - *Operador D.* La supresión de tuplas de una relación se realiza de forma similar a la de una consulta. La principal diferencia es la utilización del operador D. (suprimir tuplas) en lugar del operador P. (presentar tuplas). En QBE se pueden eliminar tuplas completas, situando el operador D. en la columna del nombre de la relación, y valores de las columnas seleccionadas, situando el operador D. en las correspondientes columnas. En este último caso, los valores eliminados se representan mediante un guión.

  - *Operador I.* Para insertar nuevos valores en una relación se puede especificar la tupla que se quiere insertar o escribir una consulta que tenga como resultado el conjunto de tuplas que se quieren insertar. La inserción se consigue situando el operador I. en la columna del nombre de la relación donde se quiere realizar esta operación.

  - *Operador U.* Este operador permite modificar un valor en una tupla. Las tuplas que se van a actualizar se pueden seleccionar mediante una consulta. En QBE el usuario no puede actualizar los campos de las claves primarias. Según el tipo de actualización, el operador U. puede aparecer en la columna del nombre de la relación o en las columnas correspondientes a los atributos.

6.4. Problemas resueltos

En esta sección se proponen y resuelven problemas utilizando QBE.

Mientras no se diga lo contrario, en los problemas se hace referencia a la base de datos AUTOMÓVILES descrita en el apéndice A.

Las tablas esqueleto correspondientes a esta base de datos son las indicas en la sección 6.2.

Como QBE esta relacionado con el cálculo relacional de dominios, para poner de manifiesto la similitud entre ellos, en algunos problemas se presentan sus correspondientes consultas utilizando las tablas esqueleto y se indica su expresión equivalente en cálculo relacional de dominios.

♦ Problema 6.1

Encontrar los nombres de los clientes que residen en 'Madrid'.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
		P._x		Madrid

Esta consulta hace que el sistema busque en la relación CLIENTES las tuplas que tienen el valor 'Madrid' en el atributo *ciudad*, asignando a la variable x el valor del atributo *nombre*, para dichas tuplas, y presentando los distintos valores que va tomando x.

Sean las variables a, x, c y d definidas en los dominios de los atributos *dni*, *CLIENTES.nombre*, *apellido* y *CLIENTES.ciudad*, respectivamente. La consulta anterior equivale en cálculo relacional de dominios a la expresión:

$$\{< x > \mid \exists\, a, c, d\ (< a, x, c, d > \in \text{CLIENTES} \wedge d = \text{'Madrid'})\}$$

Como una posición en blanco en una columna indica que su variable de dominio es única (sólo aparece una vez en la consulta), se puede eliminar la variable x de la tabla y expresar la consulta en la forma:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
		P.		Madrid

Siendo el resultado de la consulta:

nombre
Luis
Juan
María

♦ Problema 6.2

Encontrar los nombres de las ciudades donde residen los clientes cuyo apellido es 'García'.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
			García	P.

Esta consulta hace que el sistema busque en la relación CLIENTES las tuplas que tengan el valor 'García' en el atributo *apellido* y presenta los distintos valores que toma el atributo *ciudad* para dichas tuplas eliminando los duplicados automáticamente.

Sean las variables a, b, c y x definidas en los dominios de los atributos *dni*, *CLIENTES.nombre*, *apellido* y *CLIENTES.ciudad*, respectivamente. La consulta anterior equivale en cálculo relacional de dominios a la expresión:

$$\{< x > \mid \exists\, a, b, c\ (< a, b, c, x > \in \text{CLIENTES} \wedge c = \text{'García'})\}$$

ciudad
Madrid

La siguiente consulta es similar a la anterior pero utilizando la orden ALL. para no eliminar los duplicados.

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
		García		P.ALL.

ciudad
Madrid
Madrid

♦ Problema 6.3

Encontrar todas las tuplas de la relación CLIENTES.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	P._a	P._b	P._c	P._d

Sean las variables a, b, c y d definidas en los dominios de los atributos *dni*, *CLIENTES.nombre*, *apellido* y *CLIENTES.ciudad*, respectivamente. La consulta anterior equivale en cálculo relacional de dominios a la expresión:

$$\{< a, b, c, d > \mid < a, b, c, d > \in \text{CLIENTES}\}$$

Como las variables de dominio que aparecen explícitamente en la consulta son únicas para cada atributo, se pueden eliminar de la tabla y plantear la consulta en la forma:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	P.	P.	P.	P.

Por otra parte, como se quieren presentar todos los campos de la relación CLIENTES, se puede utilizar como alternativa la orden P. situado únicamente en la columna correspondiente al nombre de la relación.

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
P.				

dni	*nombre*	*apellido*	*Ciudad*
0001	Luis	García	Madrid
0002	Antonio	López	Valencia
0003	Juan	Martín	Madrid
0004	María	García	Madrid
0005	Javier	González	Barcelona
0006	Ana	López	Barcelona

♦ Problema 6.4

Obtener todas las tuplas de la relación DISTRIBUCION cuyo valor en el atributo *cantidad* sea mayor que 10.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
P.			> 10

Esta consulta hace que el sistema busque en la relación DISTRIBUCION las tuplas que tengan un valor mayor que 10 en el atributo *cantidad* y las presenta. Como se quieren todos los campos de la relación DISTRIBUCION que verifiquen la condición, se ha puesto la orden P. en la columna correspondiente al nombre de la relación.

Si se consideran las variables a, b y c definidas en los dominios de los atributos *cifc*, *dni*, y *cantidad*, respectivamente. La consulta anterior equivale en cálculo relacional de dominios a la expresión:

$$\{< a, b, c > \mid < a, b, c > \in \text{DISTRIBUCION} \wedge c > 10\}$$

cifc	*codcoche*	*cantidad*
0005	0016	20

♦ Problema 6.5

Obtener los valores del atributo *cifc* que aparecen junto a un valor del atributo *cantidad* mayor que 10 en las tuplas de la relación DISTRIBUCION.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
	P.		> 10

El problema es similar al anterior salvo en que sólo se quieren los valores del atributo *cifc*. Para ello se ha puesto la orden P. en la columna correspondiente a *cifc*.

La consulta anterior equivale en cálculo relacional de dominios a:

$$\{< a > \mid \exists\, b, c\ (< a, b, c > \in \text{DISTRIBUCION} \wedge c > 10)\}$$

cifc
0005

♦ Problema 6.6

Para el concesionario con *cifc* igual '0005', obtener los valores del atributo *codcoche* que en las tuplas de la relación DISTRIBUCION aparecen junto a un valor del atributo *cantidad* mayor que 10.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
	= 0005	P.	> 10

Este problema difiere del anterior en que en lugar de los valores del atributo *cifc* se quieren los del atributo *codcoche* imponiéndose además una nueva condición sobre *cifc*.

La consulta anterior equivale en cálculo relacional de dominios a:

$$\{< b > \mid \exists\, a, c\ (< a, b, c > \in \text{distribucion} \wedge a = 0005 \wedge c > 10)\}$$

codcoche
0016

♦ Problema 6.7

Encontrar los valores del atributo *dni* para los clientes que han comprado al menos un coche de color 'rojo' y otro coche de color 'blanco'.

Solución:

VENTAS	*cifc*	*dni*	*codcoche*	*color*
		P._x _x		rojo blanco

En esta consulta el sistema encuentra dos tuplas distintas en la relación VENTAS que coinciden en el atributo *dni*, siendo el valor para el atributo *color* el 'rojo' para una tupla y el 'blanco' para la otra. Como la orden P. se ha situado únicamente en el atributo *dni* será éste el que se presente.

La consulta anterior equivale en cálculo relacional de dominios a:

$$\{< x > \mid \exists\, a, c, d\ (< a, x, c, d > \in \text{VENTAS} \wedge d = \text{'rojo'})$$

$$\wedge \exists\, aa, cc, dd\ (< aa, x, cc, dd > \in \text{VENTAS} \wedge dd = \text{'blanco'})\}$$

dni
0001

♦ Problema 6.8

Obtener los valores del atributo *dni* para los clientes que han comprado coches de color 'rojo' o de color 'blanco' o de ambos colores.

Solución:

VENTAS	*cifc*	*dni*	*codcoche*	*color*
		P._x P._y		rojo blanco

En esta consulta el sistema encuentra las tuplas de la relación VENTAS con valor en el atributo *color* igual a 'rojo' o igual a 'blanco'. Como la orden P. se ha situado únicamente en el atributo *dni* será éste el que se presente. Obsérvese como en este problema, a diferencia del anterior, se han utilizado dos variables de dominio para el atributo *dni* en lugar de una.

La consulta anterior equivale en cálculo relacional de dominios a:

$$\{< x > \mid \exists\, a, c, d\ (< a, x, c, d > \in \text{VENTAS} \wedge (d = \text{'rojo'} \vee d = \text{'blanco'}))\}$$

o bien:

$$\{< y > \mid \exists\, a, c, d\ (< a, y, c, d > \in \text{VENTAS} \wedge (d = \text{'rojo'} \vee d = \text{'blanco'}))\}$$

dni
0001
0002
0003
0004

♦ Problema 6.9

Encontrar los valores del atributo *dni* para los clientes que han comprado coches de alguno de los colores de los coches adquiridos por el cliente con *dni* igual a '0001'.

Solución:

VENTAS	*cifc*	*dni*	*codcoche*	*color*
		0001		_x
		P._y		_x

En esta consulta, para cada tupla de la relación VENTAS con valor en el atributo *dni* igual a '0001', el sistema busca las tuplas que en dicha relación coinciden con su valor en el atributo *color*. Como la orden P. se encuentra únicamente en el atributo *dni* será éste el que se presente.

La consulta anterior equivale en cálculo relacional de dominios a:

$$\{< y > \mid \exists\, a, c, x\ (< a, y, c, x > \in \text{VENTAS} \wedge x = 0001$$

$$\wedge \exists\, aa, yy, cc\ (< aa, yy, cc, x > \in \text{VENTAS}))\}$$

dni
0001
0002
0003
0004

♦ Problema 6.10

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo'.

Solución:

VENTAS	*cifc*	*dni*	*codcoche*	*color*
		_x		rojo

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	P._x			P._y

En este caso es preciso utilizar las relaciones VENTAS y CLIENTES. Al realizar la consulta, el sistema encuentra en la relación VENTAS las tuplas con valor en el atributo *color* igual a 'rojo' y, para cada una de estas tuplas, el sistema encuentra en la relación CLIENTES las tuplas con el mismo valor en el atributo *dni* que la tupla de la relación VENTAS. Como la orden P. se encuentra en los atributos *dni* y *ciudad* serán éstos los que se presenten.

La consulta anterior equivale en cálculo relacional de dominios a:

$\{< x, y > \mid \exists\, a, c, d\ (< a, x, c, d > \in \text{VENTAS} \wedge d = \text{'rojo'})$

$\wedge \exists\, e, f\ (< x, e, f, y > \in \text{CLIENTES})\}$

dni	*ciudad*
0002	Valencia
0001	Madrid
0004	Madrid

♦ Problema 6.11

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo' o de color 'blanco' o de ambos colores.

Solución:

VENTAS	*cifc*	*dni*	*codcoche*	*color*
		_x _y		rojo blanco

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	P._x P._y			P._z P._w

El problema es similar al anterior salvo en que el color del coche puede ser 'rojo' o 'blanco'.

La consulta en QBE equivale en cálculo relacional de dominios a:

$$\{< x, z > \mid \exists\, a, c, d\ (< a, x, c, d > \in \text{VENTAS} \wedge (d = \text{'rojo'} \vee d = \text{'blanco'}))$$

$$\wedge \exists\, e, f\ (< x, e, f, z > \in \text{CLIENTES})\}$$

o bien:

$$\{< y, w > \mid \exists\, a, c, d\ (< a, y, c, d > \in \text{VENTAS} \wedge (d = \text{'rojo'} \vee d = \text{'blanco'}))$$

$$\wedge \exists\, e, f\ (< x, e, f, w > \in \text{CLIENTES})\}$$

dni	*ciudad*
0001	Madrid
0002	Valencia
0003	Madrid
0004	Madrid

♦ Problema 6.12

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado al menos un coche de color 'rojo' y otro coche de color 'blanco'.

Solución:

VENTAS	*cifc*	*dni*	*codcoche*	*color*
		_x _x		rojo blanco

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	P._x			P._z

Este problema difiere del anterior en que el los clientes deben haber adquirido al menos un coche 'rojo' y otro 'blanco'.

La consulta en QBE equivale en cálculo relacional de dominios a:

$\{< x, z > | \exists a, c, d (< a, x, c, d > \in VENTAS \wedge d = 'rojo')$

$\wedge \exists aa, cc, dd (< aa, x, cc, dd > \in VENTAS \wedge dd = 'blanco')$

$\wedge \exists e, f (< x, e, f, z > \in CLIENTES)\}$

dni	*ciudad*
0001	Madrid

♦ Problema 6.13

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo' y alguno que no sea de color 'blanco'.

Solución:

VENTAS	*cifc*	*dni*	*codcoche*	*color*
		_x		rojo
		_x		¬ blanco

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	P._x			P._z

El problema es similar al anterior imponiendo la condición de que los clientes deben haber adquirido al menos un coche 'rojo' y alguno que no sea de color 'blanco'.

dni	*ciudad*

♦ Problema 6.14

Encontrar los nombres de los coches que no tienen modelo 'gti'.

Soluciuón:

COCHES	*codcoche*	*nombre*	*modelo*
¬		P._x	gti

En la consulta, la negación se ha puesto debajo del nombre de la relación. El uso de ¬ en debajo del nombre de la relación significa *no existe*.

La consulta en QBE equivale en cálculo relacional de dominios a:

$$\{< x > \mid \neg \exists\, a, c\, (< a, x, c > \in \text{COCHES} \wedge c = \text{'gti'})\}$$

nombre
toledo
laguna
zx
xantia
a4
corsa
300
500
600

♦ Problema 6.15

Encontrar los nombres de los coches que aparecen por lo menos en dos tuplas de la relación COCHES.

Solución:

COCHES	*codcoche*	*nombre*	*modelo*
	_y ¬ _y	P._x _x	

nombre
ibiza
megane
laguna
zx
a4
astra

♦ Problema 6.16

Encontrar los nombres de los coches que aparecen por lo menos en dos tuplas de la relación COCHES y son distintos de 'astra'.

Solución:

COCHES	*codcoche*	*nombre*	*modelo*
	_y ¬ _y	P._x _x	

Condiciones
_x ¬ = astra

El cuadro de condiciones se utiliza para imponer la condición de que los nombres sean distintos de 'astra'.

nombre
ibiza
megane
laguna
zx
a4

♦ Problema 6.17

Encontrar los valores del atributo *codcoche* que aparecen en las tuplas de la relación DISTRIBUCION junto a una valor en el atributo *cantidad* mayor que 5 y menor que 10.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
		P.	_x

Condiciones
_x > 5
_x < 10

El cuadro de condiciones se utiliza para imponer la condición de que el valor en el atributo cantidad sea mayor que 5 y menor que 10.

codcoche
0005
0006
0017

♦ Problema 6.18

Obtener los valores del atributo *codcoche* que aparecen en las tuplas de la relación DISTRIBUCION junto a una valor en el atributo *cantidad* mayor que cualquiera de los correspondientes al concesionario con *cifc* igual a '0002'.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
	0002	P.	_x _y

Condiciones
_x > _y

codcoche
0005
0006
0008
0009

(continuación)

codcoche
0013
0015
0016
0017

♦ Problema 6.19

Obtener los valores del atributo *codcoche* que aparecen en las tuplas de la relación DISTRIBUCION junto a una valor en el atributo *cantidad* mayor que el doble de cualquiera de los correspondientes al concesionario con *cifc* igual a '0002'.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
	0002	P.	_x _y

Condiciones
_x > 2*_y

codcoche
0016
0006
0008
0009

♦ Problema 6.20

Encontrar los valores del atributo *codcoche* que aparecen en las tuplas de la relación DISTRIBUCION junto a una valor en el atributo *cantidad* mayor que 5 y menor que 10 pero distinto de 7.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
		P.	_x

Condiciones
_x = (> 5 and < 10 and ¬ 7)

codcoche
0017

♦ Problema 6.21

Encontrar los valores del atributo *dni* de los clientes de 'Madrid' o 'Barcelona'.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	P.			_x

Condiciones
_x = (Madrid or Barcelona)

dni
0001
0003
0004
0005
0006

♦ Problema 6.22

Obtener las tuplas de la relación MARCAS cuyo valor en el atributo *ciudad* sea igual a 'Barcelona'.

Solución:

MARCAS	*cifm*	*nombre*	*ciudad*
P.			Barcelona

cifm	*nombre*	*ciudad*
0002	renault	Barcelona
0006	bmw	Barcelona

♦ Problema 6.23

Obtener las tuplas de la relación CLIENTES cuyos valores en los atributos *apellido* y *ciudad* sean 'García' y 'Madrid', respectivamente.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
P.			García	Madrid

dni	*nombre*	*apellido*	*Ciudad*
0001	Luis	García	Barcelona
0004	María	García	Barcelona

♦ Problema 6.24

Obtener la proyección de la relación CLIENTES según el atributo *apellido*.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
			P.	

Como se indicó en álgebra relacional, la proyección de la relación CLIENTES según el atributo *apellido*, es otra relación en cuya cabecera sólo aparece el atributo especificado y cuyo cuerpo está formado por los distintos valores que toma la relación CLIENTES en dicho atributo.

apellido
García
López
Martín
Gonzalez

♦ Problema 6.25

Obtener la proyección de la relación CLIENTES según los atributos *apellido* y *ciudad*.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
			P.	P.

apellido	*ciudad*
García	Madrid
López	Valencia
Martín	Madrid
García	Madrid
González	Barcelona
López	Barcelona

♦ Problema 6.26

Seleccionar las tuplas de la relación CLIENTES cuyo valor en el atributo *ciudad* sea igual a 'Madrid' y proyectar la relación resultante según los atributos *apellido* y *ciudad*.

Solución

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
			P.	P.Madrid

apellido	*ciudad*
García	Madrid
Martín	Madrid

♦ Problema 6.27

Obtener la reunión natural de las relaciones MARCAS y CLIENTES por el atributo común *ciudad*.

Solución:

En este caso, la relación resultante es una en cuya cabecera aparecen los atributos de las relaciones MARCAS y CLIENTES, con un campo común para el nombre de la ciudad, estando su cuerpo formado por todas las posibles combinaciones de tuplas, una de cada una de las dos relaciones, tales que las dos tuplas participantes en una combinación dada correspondan a la misma ciudad.

Cuando, como en este caso, el resultado de una consulta incluye atributos de varias tablas esqueleto, se necesita un mecanismo para presentar el resultado en una sola tabla. Para ello, se declara una relación temporal RESULTADO que incluye todos los atributos del resultado de la consulta.

La presentación del resultado deseado se realiza situando la orden P. solamente en la tabla esqueleto de la relación RESULTADO.

MARCAS	*cifm*	*nombre*	*ciudad*
	_z	_y	_x

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	_c	_b	_a	_x

RESULTADO	*cifm*	*mnombre*	*dni*	*cnombre*	*apellido*	*ciudad*
P.	_z	_y	_c	_d	_a	_x

Los atributos *mnombre* y *cnombre* corresponden a los atributos *nombre* de las relaciones MARCAS y CLIENTES, respectivamente.

cifm	*mnombre*	*dni*	*cnombre*	*apellido*	*ciudad*
0001	seat	0001	Luis	García	Madrid
0001	seat	0003	Juan	Martín	Madrid
0001	seat	0004	María	García	Madrid
0002	renault	0005	Javier	González	Barcelona
0002	renault	0006	Ana	López	Barcelona
0003	citroen	0002	Antonio	López	Valencia
0004	audi	0001	Luis	García	Madrid
0004	audi	0003	Juan	Martín	Madrid
0004	audi	0004	María	García	Madrid
0006	bmw	0005	Javier	González	Barcelona
0006	bmw	0006	Ana	López	Barcelona

♦ Problema 6.28

Obtener la reunión natural de las relaciones MARCAS y CLIENTES, por el atributo común *ciudad*, y proyectar la relación resultante según los atributos *cifm*, *ciudad* y *dni*.

Solución:

Este problema es similar al anterior salvo en que la relación RESULTADO tiene por cabecera la formada por el atributo *cifm* de la relación MARCAS, el atributo *dni* de la relación CLIENTES y el atributo común *ciudad*.

MARCAS	*cifm*	*nombre*	*ciudad*
	_y		_x

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	_z			_x

RESULTADO	*cifm*	*dni*	*ciudad*
P.	_y	_z	_x

cifm	*dni*	*ciudad*
0001	0001	Madrid
0001	0003	Madrid
0001	0004	Madrid
0002	0005	Barcelona
0002	0006	Barcelona
0003	0002	Valencia
0004	0001	Madrid
0004	0003	Madrid
0004	0004	Madrid
0006	0005	Barcelona
0006	0006	Barcelona

♦ Problema 6.29

Obtener la reunión natural de las relaciones MARCAS y CLIENTES, por el atributo común *ciudad*, y proyectar la relación resultante según los atributos *cifm*, *ciudad* y *dni*, cuando el nombre de la marca es 'seat'.

Solución:

MARCAS	*cifm*	*nombre*	*ciudad*
	_y	seat	_x

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	_z			_x

RESULTADO	*cifm*	*dni*	*ciudad*
P.	_y	_z	_x

Este problema difiere del anterior en que sólo se quieren las tuplas para las que el nombre de la marca de coches es 'seat'.

cifm	*dni*	*ciudad*
0001	0001	Madrid
0001	0003	Madrid
0001	0004	Madrid

♦ Problema 6.30

Obtener los nombres de las marcas que tienen el modelo 'gtd'.

Solución:

Para resolver el problema hay que utilizar las relaciones COCHES, MARCO y MARCAS.

COCHES	*codcoche*	*nombre*	*modelo*
	_x		gtd

MARCO	*cifm*	*codcoche*
	_y	_x

MARCAS	*cifm*	*nombre*	*ciudad*
	_y	p._z	

nombre
seat
renault
citroen

♦ Problema 6.31

Obtener los nombres de las marcas de las que se han vendido coches de color 'rojo'.

Solución:

Para resolver el problema hay que utilizar las relaciones VENTAS, MARCO y MARCAS.

VENTAS	*cifc*	*dni*	*codcoche*	*color*
			_x	rojo

MARCO	*cifm*	*codcoche*
	_y	_x

MARCAS	*cifm*	*nombre*	*ciudad*
	_y	P._z	

nombre
seat
renault
citroen

♦ Problema 6.32

Obtener una relación con la suma de todos los coches existentes en el concesionario con *cifc* igual a '0001'.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
	0001		P.SUM.ALL.

Para calcular la suma de los valores indicados se utiliza el operador de agregados SUM. seguido del operador ALL. para asegurar que no se eliminan los duplicados.

sumcantidad
17

♦ Problema 6.33

Obtener el número total de nombres de coches.

Solución:

COCHES	*codcoche*	*nombre*	*modelo*
		P.CNT.UNQ.ALL	

Como todas las operaciones de agregación deben terminar con ALL., para realizar la cuenta se utiliza el operador de agregados CNT. seguido del operador UNQ. para asegurar que se eliminan los duplicados.

cntnombre
13

♦ Problema 6.34

Obtener una relación en la que aparezcan los distintos valores que toma el atributo *cifc* en la relación DISTRIBUCION junto con la suma de los valores del atributo *cantidad* para cada valor del atributo *cifc*.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
	P.G.		P.SUM.ALL._x

La orde G. se utiliza para calcular las sumas de valores del atributo *cantidad* por grupos según el atributo *cifc*.

cifc	*scantidad*
0001	17
0002	25
0003	13
0004	15
0005	38

♦ Problema 6.35

Obtener los nombres de todos los clientes en orden alfabético ascendente.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
		P.AO.		

El operador AO. se utiliza para ordenar los valores del atributo *nombre* de la relación CLIENTES de forma ascendente.

nombre
Ana
Antonio
Javier
Juan
Luis
María

♦ Problema 6.36

Obtener los nombres de los clientes de 'Madrid' en orden alfabético descendente.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
		P.DO.		Madrid

El problema es similar al anterior presentándose únicamente los clientes de 'Madrid' e invirtiendo el orden. Para ordenar los nombres de forma descendente se utiliza el operador DO.

nombre
María
Luis
Juan

♦ Problema 6.37

Obtener el nombre y el apellido de los clientes de 'Madrid' en orden alfabético ascendente por apellido y descendente por nombre.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
		P.DO(2)	P.AO(1)	Madrid

En este caso la ordenación se debe realizar por dos atributos primero por apellido AO(1) y a continuación por nombre DO(2).

nombre	*Apellido*
María	García
Luis	García
Juan	Martín

♦ Problema 6.38

Obtener los valores del atributo *dni* para los clientes cuya ciudad sea la última de la lista alfabética de las ciudades donde hay concesionarios.

Solución:

CONCESIONARIOS	*cifc*	*nombre*	*ciudad*
			_x

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	P._y			MAX.All._x

dni
0002

♦ Problema 6.39

Obtener el número total de nombres de marcas de 'Madrid'.

Solución:

MARCAS	*cifm*	*nombre*	*ciudad*
		P.CNT.ALL.	Madrid

cntnombre
2

♦ Problema 6.40

Obtener la media de los valores del atributo *cantidad* que aparecen en la relación DISTRIBUCION.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
			P.AVG.ALL.

avgcantidad
7.7

♦ Problema 6.41

Para cada valor del atributo *cifc*, obtener la media de los valores del atributo *cantidad* que aparecen en la relación DISTRIBUCION.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
	P.G.		P.AVG.ALL.

Para calcular la media de valores del atributo *cantidad* por grupos según el atributo *cifc* se utiliza el operador G.

cifc	*avgcantidad*
0001	5.7
0002	8.3
0003	4.3
0004	7.5
0005	12.7

♦ Problema 6.42

Para cada valor del atributo *cifc*, obtener la media de los valores del atributo *cantidad* que aparecen en la relación DISTRIBUCION que resulten ser mayores que 5.

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*Cantidad*
	P.G.		P.AVG.ALL._x

Condiciones
AVG.ALL._x > 5

cifc	*avgcantidad*
0001	5.7
0002	8.3
0004	7.5
0005	12.7

♦ Problema 6.43

Para los clientes de 'Madrid', obtener el mayor valor del atributo *dni*.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	P.MAX.ALL.			Madrid

maxdni
0004

♦ Problema 6.44

Para los clientes que han comprado un coche de color 'blanco', obtener el menor valor del atributo *dni*.

Solución:

VENTAS	*cifc*	*dni*	*codcoche*	*color*
		P.MIN.ALL.		blanco

mindni
0004

♦ Problema 6.45

Eliminar de la relación VENTAS la tupla que tiene los valores '0001', '0001', '0001' y 'blanco' en los atributos *cifc, dni, codcoche* y *color*, respectivamente.

Solución:

VENTAS	*cifc*	*dni*	*codcoche*	*color*
D.	0001	0001	0001	blanco

Para eliminar la tupla indicada se pone el operador D. debajo del nombre de la relación en la tabla esqueleto correspondiente a VENTAS.

♦ Problema 6.46

Eliminar de la relación VENTAS las tuplas correspondiente al concesionario con *cifc* igual a '0001'.

Solución:

VENTAS	*cifc*	*dni*	*codcoche*	*color*
D.	0001			

♦ Problema 6.47

Eliminar de la relación CLIENTES las ciudades de aquellos clientes cuyo apellido sea 'García'.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
			García	D.

En la relación resultante, las ciudades eliminadas se sustituyen por el carácter "-".

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	García	-
0002	Antonio	López	Valencia
0003	Juan	Martín	Madrid
0004	María	García	-
0005	Javier	González	Barcelona
0006	Ana	López	Barcelona

♦ Problema 6.48

Insertar en la relación VENTAS una nueva tupla en la que se refleje la venta realizada por el concesionario con *cifc* igual a '0001' al cliente con *dni* igual a '0006' de un coche con *codcoche* igual a '0005' de color 'rojo'.

Solución:

VENTAS	*cifc*	*dni*	*codcoche*	*color*
I.	0001	0006	0005	rojo

Para añadir una nueva fila en la relación VENTAS con los valores indicados se pone el operador I. debajo del nombre de la relación en la tabla esqueleto.

♦ Problema 6.49

Insertar en la relación VENTAS las tuplas que reflejen las ventas realizadas por el concesionario con *cifc* igual a '0005', a todos los clientes de 'Barcelona', de coches con *codcoche* igual a '0016' de color 'blanco'.

Solución:

En este caso se trata de insertar tuplas basadas en una consulta.

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	_x			Barcelona

VENTAS	*cifc*	*dni*	*codcoche*	*color*
I.	0005	_x	0016	blanco

♦ Problema 6.50

Actualizar la relación CLIENTES modificando la ciudad en que reside el cliente con *dni* igual a '0001' de 'Madrid' a 'Sevilla'.

Solución:

CLIENTES	*dni*	*nombre*	*apellido*	*ciudad*
	0001			U. Sevilla

Este problema se resuelve cambiando el valor del atributo *ciudad* para el cliente indicado utilizando el operador U.

♦ Problema 6.51

Actualizar la relación DISTRIBUCION aumentado en 5 unidades todos los valores del atributo *cantidad.*

Solución:

DISTRIBUCION	*cifc*	*codcoche*	*cantidad*
U.			_x + 5 _x

En este caso, cada vez que se recupera una tupla de la tabla DISTRIBUCION se determina su valor en el atributo CANTIDAD y se actualiza éste aumentándolo en 5 unidades.

Capítulo 7
QUEL

7.1 Introducción

El lenguaje de consulta QUEL se introdujo para el sistema de bases de datos *Ingres* (Interactive Graphics and Retrieval System), que se desarrolló en la Universidad de California en Berkeley. La versión original del sistema Ingres académico, sólo soportaba el lenguaje de consulta QUEL, a diferencia del sistema Ingres comercial que soportaba QUEL y SQL. Una versión comercial de Ingres fue desarrollada por *Relational Technology, Inc.(Ingres Corporation)*. La estructura básica de QUEL está estrechamente relacionada con el cálculo relacional de tuplas.

En el presente capítulo se proponen y resuelven problemas de QUEL y se realiza un recordatorio de los aspectos teóricos más importantes en relación con los problemas.

Los mandatos de QUEL pueden dividirse en dos grupos:

- *Mandatos para la definición de datos*. Son los que forman parte del lenguaje para la definición de datos DDL (Data Definition Language) de QUEL.
  - *CREATE*. Crea una tabla base
  - *INDEX*. Crea un índice
  - *DEFINE VIEW*. Crea una vista
  - *DESTROY*. Elimina una tabla base, un índice o una vista
  - *MODIFY*. Modifica una tabla base o un índice
- *Mandatos para la modificación de la base de datos y consulta*. Son los que forman parte del lenguaje de manipulación de datos DML (Data Manipulation Language) de QUEL.

- *RETRIEVE*. Recuperar
- *REPLACE*. Reemplazar
- *DELETE*. Eliminar
- *APPEND*. Añadir

7.2 Tipos de datos

Los principales tipos de datos escalares manejados por QUEL son:

- *Datos numéricos*:
  - *I1, I2, I4*. Enteros binarios de 1, 2 y 4 bytes, respectivamente
  - *F4, F8*. Números de punto flotante de 4 y 8 bytes, respectivamente
- *Datos de cadena*:
  - *CHAR (n)*. Cadena de caracteres de longitud fija n.
  - *VARCHAR (n)*. Cadena de caracteres de longitud variable hasta n
- *Datos de fecha/hora*:
  - *DATA*. Fecha y hora en valor absoluto o en un intervalo

7.3 Definición de datos

En esta sección se presenta la sintaxis de los mandatos para la definición de datos.

- *Creación de una tabla*. El comando *CREATE* de QUEL permite crear las tablas que componen la base de datos. Su sintaxis es la siguiente:

  CREATE tabla_base (definición_de_columna [, definición_de_ columna]...)

  donde *definición_de_columna* tiene la forma:

  columna = tipo_de_datos [NOT NULL]

- *Creación de un índice*. El comando *INDEX* de QUEL permite crear índices adicionales para una tabla base, además del índice que ya forma parte (si existe) de la estructura principal de dicha tabla base. Un índice creado mediante *INDEX* es un tipo especial de tabla. Su sintaxis es la siguiente:

  INDEX ON tabla_base IS (columna [, columna]...)

- *Creación de una vista.* El comando *DEFINE VIEW* de QUEL permite crear una vista de una base de datos. En QUEL sólo se debe especificar explícitamente nuevos nombres para las columnas de la vista, cuando no existe un nombre que pueda heredarse o cuando exista ambigüedad si no se hace. Su sintaxis es la siguiente:

  DEFINE VIEW vista [(lista_objetivo)] [WHERE condición]

- *Eliminación de una tabla base, un índice o una vista.* El comando *DESTROY* de QUEL permite eliminar una tabla base, un índice o una vista. Con este comando se eliminan también, de forma automática, todas las vistas o índices sobre la tabla eliminada. Su sintaxis es la siguiente:

  DESTROY tabla [, tabla]...

- *Modificación una tabla base o un índice.* El comando *MODIFY* de QUEL permite cambiar la estructura de almacenamiento de una tabla base o de un índice, es decir, permite reorganizar la tabla base o el índice en el medio de almacenamiento. Su sintaxis es la siguiente:

  MODIFY tabla TO estructure [UNIQUE] [ON columna [, columna]...]

  donde *tabla* indica una tabla base o un índice y *estructura* es una de las siguientes (el prefijo *C* opcional especifica una compresión de los datos en el disco):

  - *BTREE, CBTREE.* Arbol B
  - *HASH, CHASH.* Dispersión
  - *ISAM, CISAM.* Método de acceso secuencial indizado
  - *HEAP, CHEAP.* Cúmulo
  - *HEAPSORT, CHEAPSORT.* Cúmulo ordenado

7.4 Consultas en QUEL

En QUEL la mayor parte de las consultas se expresan mediante las cláusulas *RANGE OF*, *RETRIEVE* y *WHERE*, que se describen a continuación.

- *RANGE OF.* Permite declarar variables de tuplas. Mediante la expresión:

  RANGE OF t IS r

  se declara que *t* es una variable de tupla restringida a tomar los valores de las tuplas de la relación *r*.

- *RETRIEVE*. Su función es similar a la que tiene la cláusula *SELECT* en SQL. Esta última corresponde a la operación de *proyección* del álgebra relacional y se utiliza para listar los atributos que se desean en el resultado de una consulta.
- *WHERE*. Permite declarar el predicado de la consulta. Con la expresión:

  WHERE P

  se indica el predicado *P* que deben cumplir las tuplas que aparecen en la relación resultado de la consulta.

Una consulta en QUEL, que utiliza las cláusulas descritas, es de la forma:

RANGE OF t_1 IS r_1
RANGE OF t_2 IS r_2
.....
RANGE OF t_m IS r_m
RETRIEVE ($t_{i1}.A_{j1}$, $t_{i2}.A_{j2}$, ..., $t_{in}.A_{jn}$)
WHERE *P*

donde las t_i son variables de tupla, las r_i son relaciones y las A_{jk} son atributos. Con $t.A$ se expresa el valor de la tupla t en el atributo A.

Cuando en QUEL se realiza una consulta de la forma indicada, no se eliminan los duplicados. Para que en la relación resultante de la consulta no aparezcan duplicados, se debe añadir la palabra clave *UNIQUE* a la cláusula *RETRIEVE*. En tal caso, la consulta se expresa como sigue:

RANGE OF t_1 IS r_1
RANGE OF t_2 IS r_2
.....
RANGE OF t_m IS r_m
RETRIEVE UNIQUE ($t_{i1}.A_{j1}$, $t_{i2}.A_{j2}$, ..., $t_{in}.A_{jn}$)
WHERE *P*

Una particularidad de QUEL es que no permite subconsultas anidadas, es decir, no se puede tener una cláusula *RETRIEVE - WHERE* dentro de una cláusula *WHERE*.

En algunas consultas es posible que dos variables de tupla distintas tengan como rango la misma relación. Si para realizar la consulta sólo se precisa una variable de tupla para una misma relación, se puede omitir la sentencia *RANGE OF* para esa relación y usar el nombre de la relación como una variable de tupla declarada implícitamente. En este caso la consulta sería de la forma siguiente.

RETRIEVE ($r_{i1}.A_{j1}$, $r_{i2}.A_{j2}$, ..., $r_{in}.A_{jn}$)
WHERE *P*

Se debe resaltar que en el QUEL académico no está permitido el uso de variables de tupla declaradas implícitamente.

Para que el resultado de una consulta aparezca ordenado según uno o varios campos, se puede añadir la cláusula *SORT BY* después de la cláusula *WHERE*. En la ordenación realizada mediante *SORT BY* no aparecen duplicados.

A modo de resumen se indica a continuación la sintaxis general para realizar una consulta:

RANGE OF t_1 IS r_1
RANGE OF t_2 IS r_2
.....
RANGE OF t_m IS r_m
RETRIEVE [UNIQUE] (lista_de_asignaciones)
[WHERE condición]
[SORT BY campos]

7.5 Operaciones permitidas en QUEL

En esta sección se describen los operadores que pueden aparecer en las expresiones de QUEL.

- *Operadores de comparación* (>, <, =, ...)
- *Operadores aritméticos* (+, -, *, /)
- *Funciones integradas* (sin, cos, sqrt, interval, ...)
- *Operadores lógicas* (and, or, not)
- *Operadores de agregados*. Los operadores de agregados en QUEL se aplican sobre valores de algún atributo de una relación. El agrupamiento de valores a los que afecta el operador, se especifica como parte de la expresión de agregados. Una expresión de agregados puede tomar una de las siguientes formas:
  - *operador de agregados* (*t.A*). El operador de agregados se aplica sobre los valores de la variable de tupla *t* en el atributo *A*.
  - *operador de agregados* (*t.A* WHERE *P*). El operador de agregados se aplica sobre los valores de la variable de tupla *t* en el atributo *A* que verifican el predicado *P*.

- *operador de agregados* (*t.A* BY $S.B_1$, $S.B_2$, ..., $S.B_n$ WHERE *P*). El operador de agregados se aplica sobre los valores de la variable de tupla *t* en el atributo *A*, en grupos de tuplas $S.B_i$, que verifican el predicado *P*.

Los operadores de agregados que ofrece QUEL son:

- *COUNT, COUNTU.* Cuenta
- *SUM, SUMU.* Suma
- *AVG, AVGU.* Promedio
- *MAX.* Máximo
- *MIN.* Mínimo
- *ANY.* Existe

donde los operadores que terminan en "U" eliminan duplicados de sus operandos.

7.6 Modificación de la base de datos

En esta sección se indica la sintaxis de los mandatos *REPLACE*, *DELETE* y *APPEND*, que permiten modificar el estado de la base de datos.

- *Actualización.* En QUEL, la actualización de valores de una base de datos se realiza mediante la orden *REPLACE.*

  REPLACE variable_de_tupla (lista_de_asignaciones) [WHERE condición]

- *Eliminación.* Para eliminar valores de una base de datos se utiliza la orden *DELETE.*

  DELETE variable_de_tupla [WHERE condición]

- *Inserción.* En una base de datos se pueden insertar nuevos valores utilizando la orden *APPEND.*

  APPEND TO tabla (lista_de_asignaciones) [WHERE condición]

7.7 Problemas resueltos

En esta sección se proponen y resuelven problemas utilizando QUEL. Mientras no se diga lo contrario, en los problemas se hace referencia a la base de datos AUTOMÓVILES descrita en el apéndice A.

♦ Problema 7.1

Crear la tabla COCHES.

Solución

```
CREATE COCHES
        (codcoche   = I (4)          NOT NULL,
        nombre      = CHAR (10)      NOT NULL,
        modelo      = CHAR (10)      NOT NULL)
```

♦ Problema 7.2

Crear la tabla DISTRIBUCION.

Solución

```
CREATE DISTRIBUCION
        (cifc       = I (4)          NOT NULL,
        codcoche    = I (4)          NOT NULL,
        cantidad    = I (4)          NOT NULL)
```

♦ Problema 7.3

Crear un índice llamado *ICIFM* en la tabla MARCAS según el campo *cifm*.

Solución:

```
INDEX ON MARCAS IS ICIFM (cifm)
```

♦ Problema 7.4

Crear un índice llamado *ITODOS* en la tabla COCHES según todos los campos.

Solución:

```
INDEX ON COCHES IS ITODOS (codcoche, nombre, modelo)
```

♦ Problema 7.5

Crear una vista de la relación CLIENTES, denominada *MCLIENTES*, con las tuplas correspondientes a los clientes de 'Madrid'.

Solución:

```
DEFINE VIEW MCLIENTES
        (dni = CLIENTES.dni,
        nombre = CLIENTES.nombre,
        apellido = CLIENTES.apellido,
        madrid = CLIENTES.ciudad)
WHERE CLIENTES.ciudad = "Madrid"
```

En QUEL no es necesario especificar explícitamente nuevos nombres para las columnas de la vista cuando existe un nombre que puede heredarse y no exista ambigüedad al hacerlo. Así, la expresión anterior se puede poner como sigue:

```
DEFINE VIEW MCLIENTES
        (dni,
        nombre,
        apellido,
        madrid = CLIENTES.ciudad)
WHERE CLIENTES.ciudad = "Madrid"
```

dni	*nombre*	*apellido*	*madrid*
0001	Luis	García	Madrid
0003	Juan	Martín	Madrid
0004	María	García	Madrid

♦ Problema 7.6

Recuperar las tuplas de la vista *MCLIENTES* para los clientes con apellido igual a 'García'.

Solución:

```
RANGE OF t IS MCLIENTES
RETRIEVE (t.dni, t.nombre, t.apellido, t.madrid)
WHERE t.apellido = "García"
```

En esta expresión se puede sustituir (t.dni, t.nombre, t.apellido, t.madrid) por (t.ALL) para indicar que se quieren presentar todos los valores de las tuplas que cumplen la condición.

```
RANGE OF t IS MCLIENTES
RETRIEVE (t.ALL)
WHERE t.apellido = "García"
```

dni	*nombre*	*apellido*	*madrid*
0001	Luis	García	Madrid
0004	María	García	Madrid

♦ Problema 7.7

Borrar la tabla DISTRIBUCION.

Solución:

```
DESTROY DISTRIBUCION
```

♦ Problema 7.8

Borrar las tablas DISTRIBUCION y VENTAS.

Solución:

```
DESTROY DISTRIBUCION, VENTAS
```

♦ Problema 7.9

Modificar la tabla CLIENTES para que resulte ordenada de forma descendente según el campo *apellido* y de forma ascendente según el campo *nombre*.

Solución:

```
MODIFY CLIENTES TO HEAPSORT ON apellido:D, nombre:A
```

En la expresión la letra D indica que la ordenación es descendente por el atributo *apellido*, y la letra A que la ordenación es ascendente por el atributo *nombre*.

dni	nombre	apellido	madrid
0003	Juan	Martín	Madrid
0006	Ana	López	Barcelona
0002	Antonio	López	Valencia
0005	Javier	González	Barcelona
0001	Luis	García	Madrid
0004	María	García	Madrid

♦ Problema 7.10

Obtener todas las tuplas de la relación CONCESIONARIOS.

Solución:

La siguiente consulta hace que el sistema busque en la relación CONCESIONARIOS todas las tuplas y las presente.

```
RANGE OF t IS CONCESIONARIOS
RETRIEVE (t.cifc, t.nombre, t.ciudad)
```

o bien:

```
RANGE OF t IS CONCESIONARIOS
RETRIEVE (t.ALL)
```

cifc	nombre	ciudad
0001	acar	Madrid
0002	bcar	Madrid
0003	ccar	Barcelona
0004	dcar	Valencia
0005	ecar	Bilbao

♦ Problema 7.11

Obtener todas las tuplas de todos los clientes de 'Madrid'.

Solución:

En este caso al realizarse la consulta el sistema busca en la relación CLIENTES las tuplas que tengan el valor 'madrid' en el atributo *ciudad* y las presenta.

```
RANGE OF t IS CLIENTES
RETRIEVE (t.dni, t.nombre, t.apellido, t.ciudad)
WHERE t.ciudad = "Madrid"
```

o bien:

```
RANGE OF t IS CLIENTES
RETRIEVE (t.ALL)
WHERE t.ciudad = "Madrid"
```

Cuando en una consulta se requiere únicamente una variable de tupla cuyo rango sea una relación, se puede omitir la sentencia *RANGE OF* para dicha relación y utilizar el nombre de la relación como una variable de tupla declarada implícitamente. Así, la expresión anterior se puede poner de la forma siguiente:

```
RETRIEVE (CLIENTES.ALL)
WHERE CLIENTES.ciudad = "Madrid"
```

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	García	Madrid
0003	Juan	Martín	Madrid
0004	María	García	Madrid

♦ Problema 7.12

Encontrar los apellidos de los clientes de 'Madrid'.

Solución:

La siguiente consulta hace que el sistema busque en la relación CLIENTES las tuplas que tengan el valor 'Madrid' en el atributo *ciudad* y presenta los distintos valores que se encuentran en el atributo *apellido* para dichas tuplas.

```
RANGE OF t IS CLIENTES
RETRIEVE (t.apellido)
WHERE t.ciudad = "Madrid"
```

o bien:

```
RETRIEVE (CLIENTES.apellido)
WHERE CLIENTES.ciudad = "Madrid"
```

apellido
García
Martín
García

La consulta anterior no elimina duplicados. Así, cuando existan más de un cliente con el mismo apellido que residen en Madrid, dicho apellido aparecerá tantas veces en el resultado de la consulta como clientes existan. Para eliminar duplicados se debe añadir la clave *UNIQUE* a la cláusula *RETRIEVE*.

```
RANGE OF t IS CLIENTES
RETRIEVE UNIQUE (t.apellido)
WHERE t.ciudad = "Madrid"
```

o bien:

```
RETRIEVE UNIQUE (CLIENTES.apellido)
WHERE CLIENTES.ciudad = "Madrid"
```

apellido
García
Martín

♦ Problema 7.13

Encontrar los nombres de las ciudades donde residen los clientes que tienen como apellido 'García'.

Solución:

Con la siguiente consulta el sistema busca en la relación CLIENTES las tuplas que tengan el valor 'García' en el atributo *apellido* y presenta los distintos valores que se encuentran en el atributo *ciudad* para dichas tuplas. En la consulta se ha utiliza la clave *UNIQUE* para que no aparezcan las ciudades repetidas.

```
RANGE OF t IS CLIENTES
RETRIEVE UNIQUE (t.ciudad)
WHERE t.apellido = "García"
```

ciudad
Madrid

♦ Problema 7.14

Obtener las tuplas de la relación DISTRIBUCION para las que el atributo *cantidad* toma un valor mayor que 7.

Solución:

La siguiente consulta hace que el sistema busque en la relación DISTRIBUCION las tuplas que tengan un valor mayor que 7 en el atributo *cantidad* y las presenta.

```
RANGE OF t IS DISTRIBUCION
RETRIEVE (t.cifc, t.codcoche, t.cantidad)
WHERE t.cantidad > 7
```

o bien:

```
RANGE OF t IS DISTRIBUCION
RETRIEVE (t.ALL)
WHERE t.cantidad > 7
```

cifc	*codcoche*	*cantidad*
0002	0008	10
0002	0009	10
0004	0013	10
0005	0015	10
0005	0016	20
0005	0017	8

♦ Problema 7.15

Obtener las tuplas de la relación DISTRIBUCION para las que el atributo *cantidad* toma un valor mayor que 7 y ordenar el resultado de forma ascendente por el atributo *cifc* y descendente por el atributo *codcoche*.

Solución:

Este problema se resuelve como el anterior incluyendo la cláusula *SORT BY*. Para que la ordenación se realice de forma ascendente se utiliza la letra A y para que se realice de forma descendente se utiliza la letra D. Si no se indica el tipo de ordenación el sistema realiza por defecto la ascendente.

```
RANGE OF t IS DISTRIBUCION
RETRIEVE (t.cifc, t.codcoche, t.cantidad)
WHERE t.cantidad > 7
SORT BY cifc:A, codcoche:D
```

o bien:

```
RETRIEVE (DISTRIBUCION.ALL)
WHERE DISTRIBUCION.cantidad > 7
SORT BY cifc:A, codcoche:D
```

cifc	*codcoche*	*cantidad*
0002	0009	10
0002	0008	10
0004	0013	10
0005	0017	8
0005	0016	20
0005	0015	10

♦ Problema 7.16

Para las tuplas de la relación DISTRIBUCION con valores de *cifc* iguales a '0001', obtener una tabla en la que aparezcan los atributos *cifc*, *codcoche*, *comentario* y *ncantidad*, siendo los valores de la columna *comentario* iguales a 'Nueva cantidad =' y los valores de *ncantidad* los del atributo *cantidad* multiplicados por dos.

Solución:

En este caso es necesario introducir nuevos nombres para la tercera y la cuarta columna al no existir nombres heredables de las relaciones que componen la base de datos.

```
RANGE OF t IS DISTRIBUCION
RETRIEVE
        (t.cifc,
        t.codcoche,
        comentario = "Nueva cantidad =",
        ncantidad = cantidad * 2)
WHERE t.cifc = 0001
```

cifc	*codcoche*	*comentario*	*ncantidad*
0001	001	Nueva cantidad =	6
0001	005	Nueva cantidad =	14
0001	016	Nueva cantidad =	14

♦ Problema 7.17

Obtener los valores del atributo *cifc* que aparecen en las tuplas de la relación DISTRIBUCION junto a un valor en el atributo *cantidad* mayor que 7.

Solución:

En este caso sólo se quieren los valores del atributo *cifc* de la relación DISTRIBUCION que cumplen la condición. La cláusula *UNIQUE* se introduce para eliminar duplicados.

```
RANGE OF t IS DISTRIBUCIÓN
RETRIEVE UNIQUE (t.cifc)
WHERE t.cantidad > 7
```

cifc
0002
0004
0005

♦ Problema 7.18

Obtener los valores del atributo *codcoche* que aparecen en las tuplas de la relación DISTRIBUCION junto a un valor en el atributo *cantidad* mayor que 7 y junto a un valor en el atributo *cifc* igual a '0002'

Solución:

```
RANGE OF t IS DISTRIBUCIÓN
RETRIEVE UNIQUE (t.codcoche)
WHERE t.cantidad > 7 AND t.cifc = 0002
```

codcoche
0008
0009

♦ Problema 7.19

Encontrar los valores del atributo *dni* para los clientes que han comprado al menos un coche de color 'rojo' y otro coche de color 'blanco'.

Solución:

En este caso hay que utilizar dos variables de tuplas para la relación VENTAS, ya que si se plantea la consulta de la forma:

```
RANGE OF t IS VENTAS
RETRIEVE UNIQUE (t.dni)
WHERE t.color = "rojo" AND t.color = "blanco"
```

el resultado sería una tabla vacía, debido a que no es posible que en una misma tupla el atributo *color* tome simultáneamente los valores 'rojo' y 'blanco'. La forma correcta de resolver el problema es mediante la siguiente expresión, en la que no se pueden utilizar variables de tupla implícitas.

```
RANGE OF t IS VENTAS
RANGE OF s IS VENTAS
RETRIEVE UNIQUE (s.dni)
WHERE t.color = "rojo" AND s.dni = t.dni AND s.color = "blanco"
```

dni
0001

♦ Problema 7.20

Obtener los valores del atributo *dni* para los clientes que han comprado coches de color 'rojo' o de color 'blanco' o de ambos colores.

Solución:

La diferencia entre este problema y el anterior se encuentra en la condición que deben cumplir los colores de los coches. En este caso si es posible resolver el problema utilizando una sola variable de tupla, ya que la condición que se impone es que el valor del atributo *color* en cada tupla sea 'rojo' o 'blanco'.

```
RANGE OF t IS VENTAS
RETRIEVE UNIQUE (t.dni)
WHERE t.color = "rojo" OR t.color = "blanco"
```

dni
0001
0002
0003
0004

♦ Problema 7.21

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo'.

Solución:

Este problema difiere de los anteriores en que es preciso utilizar dos relaciones, VENTAS y CLIENTES, en lugar de una. El proceso seguido para resolver el problema consiste en:

- Encontrar las tuplas de la relación VENTAS cuyo valor en el atributo *color* es 'rojo'.
- Seleccionar las tuplas de la relación CLIENTES para los valores de *dni* correspondientes a las tuplas obtenidas de la relación VENTAS.
- Presentar los atributos *dni* y *ciudad* de las tuplas seleccionadas de la relación CLIENTES.

```
RANGE OF t IS VENTAS
RANGE OF s IS CLIENTES
RETRIEVE UNIQUE (s.dni, s.ciudad)
WHERE t.color = "rojo" AND s.dni = t.dni
```

o bien:

```
RETRIEVE UNIQUE (CLIENTES.dni, CLIENTES.ciudad)
WHERE VENTAS.color = "rojo" AND CLIENTES.dni = VENTAS.dni
```

dni	*ciudad*
0002	Valencia
0001	Madrid
0004	Madrid

♦ Problema 7.22

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo' o de color 'blanco' o de ambos colores.

Solución:

El problema es similar al anterior salvo en que el color del coche puede ser 'rojo' o 'blanco'.

```
RANGE OF t IS VENTAS
RANGE OF s IS CLIENTES
RETRIEVE UNIQUE (s.dni, s.ciudad)
WHERE (t.color = "rojo" OR t.color = "blanco") AND s.dni = t.dni
```

o bien:

```
RETRIEVE UNIQUE (CLIENTES.dni, CLIENTES.ciudad)
WHERE (VENTAS.color = "rojo" OR VENTAS.color = "blanco")
            AND CLIENTES.dni = VENTAS.dni
```

dni	*ciudad*
0001	Madrid
0002	Valencia
0003	Madrid
0004	Madrid

♦ Problema 7.23

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado al menos un coche de color 'rojo' y otro coche de color 'blanco'.

Solución:

En este caso no se puede resolver el problema utilizando únicamente variables de tupla de forma implícita, ya que para imponer la condición de haber comprado un coche de color 'rojo' y otro de color 'blanco' se deben utilizar dos variables de tuplas para la relación VENTAS.

```
RANGE OF t IS VENTAS
RANGE OF s IS VENTAS
RANGE OF u IS CLIENTES
RETRIEVE UNIQUE (u.dni, u.ciudad)
WHERE t.color = "rojo"
            AND s.dni = t.dni
            AND s.color = "blanco"
            AND u.dni = s.dni
```

dni	*ciudad*
0001	Madrid

♦ Problema 7.24

Obtener los valores de los atributos *dni* y *ciudad* para los clientes que han comprado coches de color 'rojo' y alguno que no sea de color 'blanco'.

Solución:

Como en el problema anterior, éste no se puede resolver utilizando sólo variables de tupla de forma implícita, ya que para imponer la condición de haber comprado un coche de color 'rojo' y alguno que no sea de color 'blanco' se deben utilizar dos variables de tuplas para la relación VENTAS.

```
RANGE OF t IS VENTAS
RANGE OF s IS VENTAS
RANGE OF u IS CLIENTES
RETRIEVE UNIQUE (u.dni, u.ciudad)
WHERE t.color = "rojo"
            AND s.dni = t.dni
            AND s.color <> "blanco"
            AND u.dni = s.dni
```

dni	*ciudad*

♦ Problema 7.25

Obtener los nombres de todas las marcas de coches ordenadas alfabéticamente.

Solución:

```
RANGE OF t IS MARCAS
RETRIEVE (t.nombre)
SORT BY nombre
```

Al no especificarse el tipo de ordenación en la cláusula *SORT BY* por defecto se considera ascendente.

nombre
audi
bmw
citroen
opel
renault
seat

♦ Problema 7.26

Obtener los valores del atributo *cifc* que aparecen en las tuplas de la relación DISTRIBUCION junto a un valor en el atributo *cantidad* comprendido entre 10 y 18 ambos inclusive.

Solución:

```
RETRIEVE UNIQUE (DISTRIBUCION.cifc)
WHERE DISTRIBUCION.cantidad >= 10
            AND DISTRIBUCION.cantidad <=18
```

cifc
0002
0004
0005

♦ Problema 7.27

Obtener las tuplas de la relación DISTRIBUCION para las que el atributo *cantidad* toma un valor mayor que 10 o menor que 5.

Solución:

```
RETRIEVE UNIQUE (DISTRIBUCION.cifc)
WHERE DISTRIBUCION.cantidad > 10 OR DISTRIBUCION.cantidad < 5
```

cifc
0001
0003
0005

♦ Problema 7.28

Obtener todas las parejas de valores de los atributos *cifm* de la relación *marcas* y *dni* de la relación *clientes* que sean de la misma ciudad.

Solución:

El problema consiste en reunir las relaciones MARCAS y CLIENTES, según el atributo común *ciudad*, y a continuación presentar los valores de los atributos *cifm* y *dni* de la relación resultante.

```
RETRIEVE UNIQUE (MARCAS.cifm, CLIENTES.dni)
WHERE MARCAS.ciudad = CLIENTES.ciudad
```

cifm	*dni*
0001	0001
0001	0003
0001	0004
0002	0005
0002	0006
0003	0002
0004	0001
0004	0003
0004	0004
0006	0005
0006	0006

♦ Problema 7.29

Obtener todas las parejas de valores de los atributos *cifm* de la relación *marcas* y *dni* de la relación *clientes* que no sean de la misma ciudad.

Solución:

Este problema es similar al anterior imponiendo la condición de que la ciudad de la marca sea distinta a la del cliente.

```
RETRIEVE UNIQUE (MARCAS.cifm, CLIENTES.dni)
WHERE NOT MARCAS.ciudad = CLIENTES.ciudad
```

cifm	*dni*
0001	0002
0001	0005
0001	0006
0002	0001
0002	0002
0002	0003
0002	0004
0003	0001
0003	0003
0003	0004
0003	0005
0003	0006
0004	0002

(continuación)

cifm	*dni*
0004	0005
0004	0006
0005	0001
0005	0002
0005	0003
0005	0004
0005	0005
0005	0006
0006	0001
0006	0002
0006	0003
0006	0004

♦ Problema 7.30

Obtener los valores del atributo *codcoche* para los coches que se encuentran en algún concesionario de 'Barcelona'.

Solución:

Para resolver este problema, habrá primero que calcular el *cifc* de los concesionarios que son de Barcelona y después ver cuales de ellos se encuentran en la relación DISTRIBUCION.

```
RETRIEVE UNIQUE (DISTRIBUCION.codcoche)
WHERE CONCESIONARIOS.ciudad = "Barcelona"
        AND DISTRIBUCION.cifc = CONCESIONARIOS.cifc
```

codcoche
0010
0011
0012

♦ Problema 7.31

Obtener el valor del atributo *codcoche* de aquellos coches vendidos a clientes de 'Madrid'.

Solución:

Para resolver este problema, hay que determinar primero el *dni* de los clientes que son de 'Madrid' y después ver cuales se encuentran en la relación VENTAS.

```
RANGE OF t IS CLIENTES
RANGE OF s IS VENTAS
RETRIEVE UNIQUE (s.codcoche)
WHERE t.ciudad = "Madrid"
        AND s.dni = t.dni
```

o bien:

```
RETRIEVE UNIQUE (VENTAS.codcoche)
WHERE CLIENTES.ciudad = "Madrid"
        AND VENTAS.dni = CLIENTES.dni
```

codcoche
0001
0006
0011
0008

♦ Problema 7.32

Obtener los valores del atributo *codcoche* para los coches que han sido adquiridos por un cliente de 'Madrid' en un concesionario de 'Madrid'.

Solución:

Para resolver el problema se necesita utilizar las relaciones VENTAS, CONCESIONARIOS y CLIENTES. El proceso de resolución consiste en encontrar los valores del atributo *dni* para los clientes de 'Madrid' y los valores del atributo *cifc* para los concesionarios de 'Madrid' y obtener los valores del atributo *codcoche* que en la tabla VENTAS aparecen en tuplas junto a valores de *cifc* y *dni* de los encontrados anteriormente.

```
RANGE OF t IS VENTAS
RANGE OF s IS CONCESIONARIOS
RANGE OF u IS CLIENTES
RETRIEVE UNIQUE (t.codcoche)
WHERE s.ciudad = "Madrid"
            AND u.ciudad = "Madrid"
            AND t.cifc = s.cifc
            AND t.dni = u.dni
```

o bien:

```
RETRIEVE UNIQUE (VENTAS.codcoche)
WHERE CONCESIONARIOS.ciudad = "Madrid"
            AND CLIENTES.ciudad = "Madrid"
            AND VENTAS.cifc = CONCESIONARIOS.cifc
            AND VENTAS.dni = CLIENTES.dni
```

codcoche
0001
0008
0006

♦ Problema 7.33

Obtener los valores del atributo *codcoche* para los coches comprados en un concesionario de la misma ciudad que la del cliente que lo compra.

Solución:

La forma de resolver este problema es similar a la aplicada en el caso anterior, imponiendo que las ciudades del concesionario y del cliente sean las mismas.

```
RANGE OF t IS VENTAS
RANGE OF s IS CONCESIONARIOS
RANGE OF u IS CLIENTES
RETRIEVE UNIQUE (t.codcoche)
WHERE s.ciudad = u.ciudad
            AND t.cifc = s.cifc
            AND t.dni = u.dni
```

o bien:

```
RETRIEVE UNIQUE (VENTAS.codcoche)
WHERE CONCESIONARIOS.ciudad = CLIENTES.ciudad
            AND VENTAS.cifc = CONCESIONARIOS.cifc
            AND VENTAS.dni = CLIENTES.dni
```

codcoche
0001
0008
0006

♦ Problema 7.34

Obtener los valores del atributo *codcoche* para los coches comprados en un concesionario de distinta ciudad que la del cliente que lo compra.

Solución:

Este problema se resuelve como el anterior, imponiendo la condición de que las ciudades del concesionario y del cliente sean distintas.

```
RETRIEVE UNIQUE (VENTAS.codcoche)
WHERE CONCESIONARIOS.ciudad <> CLIENTES.ciudad
      AND VENTAS.cifc = CONCESIONARIOS.cifc
      AND VENTAS.dni = CLIENTES.dni
```

codcoche
0005
0011
0014

♦ Problema 7.35

Obtener todas las parejas de nombre de marcas que sean de la misma ciudad.

Solución:

El proceso de resolución de este problema consiste en realizar una reunión natural de la relación MARCAS consigo misma según el atributo *ciudad*, para después seleccionar las tuplas resultantes y proyectar sobre los campos nombre. El propósito de la condición t.nombre < s.nombre es eliminar las parejas (nombre1, nombre1) de nombres iguales y garantizar la aparición de una sola de las parejas (nombre1, nombre2) y (nombre2, nombre1). En este caso no es posible utilizar variables de tupla de forma implícita.

```
RANGE OF t IS MARCAS
RANGE OF s IS MARCAS
RETRIEVE (t.nombre, s.nombre)
WHERE t.ciudad = s.ciudad
                AND t.nombre < s.nombre
```

t.nombre	s.nombre
audi	seat
bmw	renault

♦ Problema 7.36

Obtener todas las parejas de modelos de coches que tengan el mismo nombre de coche y su marca sea de la ciudad de 'Bilbao'.

Solución:

Este problema se resuelve de forma similar al problema anterior utilizando las relaciones COCHES, MARCO y MARCAS.

```
RANGE OF t IS MARCAS
RANGE OF s IS MARCO
RANGE OF u IS COCHES
RANGE OF v IS COCHES
RETRIEVE (u.modelo, v.modelo)
WHERE t.ciudad = "Bilbao"
        AND s.cifm = t.cifm
        AND u.codcoche = s.codcoche
        AND v.nombre = u.nombre
        AND v.modelo < v.modelo
```

modelo	*nmodelo*
caravan	gti

♦ Problema 7.37

Obtener todos los valores del atributo *codcoche* para los coches cuyo nombre empieza por la letra 'c'.

Solución:

En QUEL se pueden utilizar los siguientes caracteres especiales para especificar consultas con correspondencia parcial en una comparación de cadenas:

- El carácter "?" corresponde a cualquier carácter individual
- El carácter "*" corresponde a cualquier secuencia de cero o más caracteres
- La cadena "[xyz]", donde xyz es cualquier conjunto de caracteres, corresponde a cualquier carácter en xyz
- Estos caracteres especiales se consideran como caracteres ordinarios, cuando van precedidos por "\"

```
RETRIEVE (COCHES.codcoche)
WHERE COCHES.nombre = "c*"
```

codcoche
0005
0017

♦ Problema 7.38

Obtener el número total de nombres de marcas domiciliadas en 'Madrid'.

Solución:

```
RANGE OF t IS MARCAS
RETRIEVE COUNTU (t.nombre WHERE t.ciudad = "MADRID")
```

En la consulta se ha utilizado *COUNTU* en lugar de *COUNT* para evitar contar los nombres de marcas repetidos.

COUNT(nombre)
2

♦ Problema 7.39

Obtener el máximo valor que toma el atributo *dni* para los clientes de 'Madrid'.

Solución:

```
RANGE OF t IS CLIENTES
RETRIEVE MAX (t.dni WHERE t.ciudad = "Madrid")
```

MAX(dni)
0004

♦ Problema 7.40

Obtener el mínimo valor que toma el atributo *dni* para los clientes que han comprado un coche de color 'blanco'.

Solución:

```
RANGE OF t IS VENTAS
RETRIEVE MIN (t.dni WHERE t.color = "BLANCO")
```

MIN(dni)
0001

♦ Problema 7.41

Obtener el *dni* de los clientes que han comprado algún coche en un concesionario de 'Madrid'.

Solución:

```
RANGE OF t IS CONCESIONARIOS
RANGE OF s IS VENTAS
RETRIEVE UNIQUE (s.dni)
WHERE t.ciudad = "Madrid" AND s.cifc = t.cifc
```

dni
0001
0002
0003

♦ Problema 7.42

Obtener los colores de los coches vendidos por el concesionario 'acar'.

Solución:

```
RANGE OF t IS CONCESIONARIOS
RANGE OF s IS VENTAS
RETRIEVE UNIQUE (s.color)
WHERE t.nombre = "acar" AND s.cifc = t.cifc
```

color
blanco
rojo

♦ Problema 7.43

Obtener los valores del atributo *codcoche* para los coches vendidos por algún concesionario de 'Madrid'.

Solución:

```
RANGE OF t IS CONCESIONARIOS
RANGE OF s IS VENTAS
RETRIEVE UNIQUE (s.codcoche)
WHERE t.ciudad = "Madrid" AND s.cifc = t.cifc
```

codcoche
001
005
008
006

♦ Problema 7.44

Obtener el nombre y el modelo de los coches vendidos por algún concesionario de 'Barcelona'.

Solución:

```
RANGE OF t IS CONCESIONARIOS
RANGE OF s IS VENTAS
RANGE OF u IS COCHES
RETRIEVE UNIQUE (u.nombre, u.modelo)
WHERE t.ciudad = "Barcelona"
            AND s.cifc = t.cifc
            AND u.codcoche = s.codcoche
```

o bien:

```
RETRIEVE UNIQUE (COCHES.nombre, COCHES.modelo)
WHERE CONCESIONARIOS.ciudad = "Barcelona"
            AND VENTAS.cifc = CONCESIONARIOS.cifc
            AND COCHES.codcoche = VENTAS.codcoche
```

nombre	*modelo*
zx	td

♦ Problema 7.45

Obtener todos los nombres de los clientes que hayan adquirido algún coche en el concesionario 'dcar'.

Solución:

```
RANGE OF t IS CONCESIONARIOS
RANGE OF s IS VENTAS
RANGE OF u IS CLIENTES
RETRIEVE UNIQUE (u.nombre)
WHERE t.nombre = "dcar" AND s.cifc = t.cifc AND u.dni = s.dni
```

nombre
Javier

♦ Problema 7.46

Obtener el nombre y el apellido de los clientes que han adquirido un coche modelo 'gti' de color 'blanco'.

Solución:

```
RANGE OF t IS CLIENTES
RANGE OF s IS VENTAS
RANGE OF u IS COCHES
RETRIEVE UNIQUE (t.nombre, t.apellido)
WHERE u.modelo = "gti"
                AND s.codcoche = u.codcoche
                AND s.color = "blanco"
                AND t.dni = s.dni
```

nombre	*apellido*

♦ Problema 7.47

Obtener el nombre y el apellido de los clientes que han adquirido un automóvil en un concesionario que dispone de coches del modelo 'gti'.

Solución:

```
RANGE OF t IS CLIENTES
RANGE OF s IS VENTAS
RANGE OF u IS DISTRIBUCION
RANGE OF v IS COCHES
RETRIEVE UNIQUE (t.nombre, t.apellido)
WHERE v.modelo = "gti" AND u.codcoche = v.codcoche
                AND s.cifc = u.cifc AND t.dni = s.dni
```

nombre	*apellido*
Luis	García
Antonio	López

♦ Problema 7.48

Obtener el nombre y el apellido de los clientes que han adquirido un automóvil en un concesionario de 'Madrid' que dispone de coches del modelo 'gti'.

Solución:

```
RANGE OF t IS COCHES
RANGE OF s IS DISTRIBUCION
RANGE OF u IS CONCESIONARIOS
RANGE OF v IS VENTAS
RANGE OF w IS CLIENTES
RETRIEVE UNIQUE (w.nombre, w.apellido)
WHERE t.modelo = "gti"
            AND s.codcoche = t.codcoche
            AND u.cifc = s.cifc
            AND u.ciudad = "Madrid"
            AND v.cifc = u.cifc
            AND w.dni = v.dni
```

o bien:

```
RETRIEVE UNIQUE (CLIENTES.nombre, CLIENTES.apellido)
WHERE COCHES.modelo = "gti"
            AND DISTRIBUCION.codcoche = COCHES.codcoche
            AND CONCESIONARIOS.cifc = DISTRIBUCION.cifc
            AND CONCESIONARIOS.ciudad = "Madrid"
            AND VENTAS.cifc = CONCESIONARIOS.cifc
            AND CLIENTES.dni = VENTAS.dni
```

nombre	*apellido*
Luis	Garcia

♦ Problema 7.49

Obtener el nombre y el apellido de los clientes cuyo *dni* es menor que el correspondiente al cliente de nombre 'Juan' y apellido 'Martín'.

Solución:

Para resolver el problema hay que utilizar dos variables de tupla para la relación CLIENTES.

```
RANGE OF t IS CLIENTES
RANGE OF s IS CLIENTES
```

```
RETRIEVE UNIQUE (s.nombre, s.apellido)
WHERE t.mombre = "Juan" AND t.apellido = "Martín" AND s.dni < t.dni
```

nombre	*apellido*
Luis	García
Antonio	López

♦ Problema 7.50

Obtener el nombre y el apellido de los clientes cuyo *dni* es menor que el de todos los clientes que son de 'Barcelona'.

Solución:

Como en el problema anterior, de nuevo hay que utilizar dos variables de tupla para la relación CLIENTES.

```
RANGE OF t IS CLIENTES
RANGE OF s IS CLIENTES
RETRIEVE UNIQUE (s.nombre, s.apellido)
WHERE s.dni < MIN (t.dni WHERE t.ciudad = "Barcelona")
```

nombre	*apellido*
Luis	García
Antonio	López
Juan	Martín
María	García

♦ Problema 7.51

Obtener el nombre y el apellido de los clientes cuyo nombre empieza por 'A' y cuyo *dni* es mayor que el de todos los clientes que son de 'Madrid'.

Solución:

```
RANGE OF t IS CLIENTES
RANGE OF s IS CLIENTES
RETRIEVE UNIQUE (s.nombre, s.apellido)
WHERE s.nombre = "A*"
            AND s.dni > MAX (t.dni WHERE t.ciudad = "Madrid")
```

nombre	*apellido*
Ana	López

♦ Problema 7.52

Obtener el nombre y el apellido de los clientes cuyo nombre empieza por 'A' y cuyo *dni* es mayor que el de alguno de los clientes que son de 'Madrid'.

Solución:

```
RANGE OF t IS CLIENTES
RANGE OF s IS CLIENTES
RETRIEVE UNIQUE (s.nombre, s.apellido)
WHERE s.nombre = "A*" AND t.ciudad = "Madrid" AND s.dni > t.dni
```

nombre	*apellido*
Antonio	López
Ana	López

♦ Problema 7.53

Obtener el nombre y el apellido de los clientes cuyo nombre empieza por 'A' y cuyo *dni* es mayor que el de alguno de los clientes que son de 'Madrid' o menor que el de todos los de 'Valencia'.

Solución:

```
RANGE OF t IS CLIENTES
RANGE OF s IS CLIENTES
RETRIEVE UNIQUE (s.nombre, s.apellido)
WHERE s.nombre = "A*"
            AND ((t.ciudad = "Madrid" AND s.dni > t.dni)
            OR (s.dni < MIN (t.dni WHERE t.ciudad = "Valencia")))
```

Obsérvese en la resolución del problema la situación de los paréntesis para poder imponer la condición OR.

nombre	*apellido*
Antonio	López
Ana	López

♦ Problema 7.54

Obtener el nombre y el apellido de los clientes que han comprado como mínimo un coche 'blanco' y un coche 'rojo'.

Solución:

```
RANGE OF t IS VENTAS
RANGE OF s IS VENTAS
RANGE OF u IS CLIENTES
RETRIEVE UNIQUE (u.nombre, u.apellido)
WHERE t.color = "rojo"
                AND s.dni = t.dni
                AND s.color = "blanco"
                AND u.dni = s.dni
```

nombre	*apellido*
Luis	García

♦ Problema 7.55

Obtener el *dni* de los clientes cuya ciudad sea la última de la lista alfabética de las ciudades donde hay concesionarios.

Solución:

```
RANGE OF t IS CONCESIONARIOS
RANGE OF s IS CLIENTES
RETRIEVE UNIQUE (s.dni)
WHERE s.ciudad = MAX (t.ciudad)
```

o bien:

```
RETRIEVE UNIQUE (CLIENTES.dni)
WHERE CLIENTES.ciudad = MAX (CONCESIONARIOS.ciudad)
```

dni
0002

♦ Problema 7.56

Obtener la media de los valores del atributo *cantidad* que aparecen en las tuplas de la relación DISTRIBUCION.

Solución:

```
RANGE OF t IS DISTRIBUCION
RETRIEVE AVG (t.cantidad)
```

AVG(cantidad)
7.7

♦ Problema 7.57

Para cada valor del atributo *cifc*, obtener la media de los valores del atributo *cantidad* que aparecen en la relación DISTRIBUCION.

Solución:

En este caso la media se calcula por grupos para los distintos valores de *cifc*.

```
RANGE OF t IS DISTRIBUCION
RETRIEVE (t.cifc, avgcantidad = AVG (t.cantidad BY t.cifc))
```

Obsérvese como en este caso se ha asignado un nombre para el resultado de las medias.

cifc	*avgcantidad*
0001	5.7
0002	8.3
0003	4.3
0004	7.5
0005	12.7

♦ Problema 7.58

Obtener los valores del atributo *cifc* que aparecen en las tuplas de la relación DISTRIBUCION junto a un valor en el atributo *cantidad* mayor que la media de todos los valores del atributo *cantidad*.

Solución:

```
RANGE OF t IS DISTRIBUCION
RANGE OF s IS DISTRIBUCION
RETRIEVE (s.cifc)
WHERE s.cantidad > AVG (t.cantidad)
```

cifc
0002
0004
0005

♦ Problema 7.59

Obtener los valores del atributo *cifc* que aparecen en las tuplas de la relación DISTRIBUCION junto a un valor en el atributo *cantidad* mayor que la media de los valores del atributo *cantidad* correspondiente al concesionario con *cifc* igual a '0001'.

Solución:

```
RANGE OF t IS DISTRIBUCION
RANGE OF s IS DISTRIBUCION
RETRIEVE (s.cifc)
WHERE s.cantidad > AVG (t.cantidad WHERE t.cifc = 0001)
```

cifc
0001
0002
0004
0005

♦ Problema 7.60

Obtener los valores del atributo *codcoche* que aparecen en las tuplas de la relación DISTRIBUCION junto a un valor en el atributo *cantidad* mayor que la media de los valores del atributo *cantidad* correspondiente al valor del atributo *cifc* que aparece junto a él.

Solución:

Para resolver el problema se deben calcular por grupos las medias de los valores del atributo *cantidad* para los distintos valores del atributo *cifc*.

```
RANGE OF t IS DISTRIBUCION
RANGE OF s IS DISTRIBUCION
RETRIEVE (s.codcoche)
WHERE s.cantidad > AVG (t.cantidad BY s.cifc WHERE t.cifc = s.cifc)
```

codcoche
0005
0006
0008
0009
0010
0012
0013
0016

♦ Problema 7.61

Utilizando el operador *ANY*, obtener los nombres de los clientes que han comprado un coche de color 'verde'.

Solución:

El operador de agregados *ANY* devuelve el valor 0 si el conjunto que constituye su argumento está vacío, y el valor 1 en caso contrario. La condición *ANY (argumento) = 1* es análoga al cuantificador "*existe*", y la condición *ANY (argumento) = 0* es análoga a "*no existe*".

```
RETRIVE (CLIENTES.nombre)
WHERE ANY (VENTAS.dni BY CLIENTES.dni
                WHERE VENTAS.color = "verde"
                AND CLIENTES.dni = VENTAS.dni) = 1
```

En la práctica no tiene mucho sentido utilizar *ANY (argumento) = 1*, pues una consulta de este tipo siempre se puede expresar de forma más sencilla sin utilizar *ANY*.

```
RETRIVE (CLIENTES.nombre)
WHERE VENTAS.color = "verde" AND CLIENTES.dni = VENTAS.dni
```

nombre
Javier

♦ Problema 7.62

Obtener los nombres de los clientes que no han comprado un coche de color 'verde'.

Solución:

Este problema se resuelve de forma similar al problema anterior cambiando *ANY (argumento) = 1* por *ANY (argumento) = 0*.

```
RETRIVE (CLIENTES.nombre)
WHERE ANY (VENTAS.dni BY CLIENTES.dni
            WHERE VENTAS.color = "verde"
            AND CLIENTES.dni = VENTAS.dni) = 0
```

En este caso la realización de la consulta sin utilizar ANY no es trivial. Obsérvese que si la consulta se plantea como se indica a continuación, no se llega al mismo resultado que con la consulta anterior, ya que al no aplicarse la condición de color por grupos, un cliente que haya comprado un coche de color verde y otro de distinto color aparecería en la tabla de resultados.

```
RETRIVE (CLIENTES.nombre)
WHERE VENTAS.color <> "verde" AND CLIENTES.dni = VENTAS.dni
```

La condición *ANY (argumento) = 0* se puede sustituir por la condición *COUNT (argumemto) = 0*, pero es preferible la primera por estar más cercana al lenguaje natural.

```
RETRIVE (CLIENTES.nombre)
WHERE COUNT (VENTAS.dni BY CLIENTES.dni
            WHERE VENTAS.color = "verde"
            AND CLIENTES.dni = VENTAS.dni) = 0
```

nombre
Luis
Antonio
Juan
María
Ana

♦ Problema 7.63

Obtener los valores del atributo *dni* para los clientes que sólo han comprado coches al concesionario con *cifc* igual a '0001'.

Solución:

Este problema se puede plantear como: "obtener los valores del atributo *dni* para los clientes que en la relación VENTAS no les corresponde ningún valor de *cifc* distinto de 0001".

```
RANGE OF t IS VENTAS
RANGE OF s IS VENTAS
RETRIVE (s.dni)
WHERE ANY (t.dni BY s.dni WHERE s.cifc <> 0001 AND s.cifc = t.cifc) = 0
```

dni
0002

♦ Problema 7.64

Obtener los nombres de los clientes que no han comprado coches de color 'rojo' en concesionarios de 'Madrid'.

Solución:

El problema se puede replantear como: "obtener el nombre de los clientes cuyo *dni* no aparece en ninguna tupla de la relación VENTAS junto al *color* 'rojo' y a un *cifc* de concesionarios de Madrid".

```
RETRIVE (CLIENTES.nombre)
WHERE ANY (VENTAS.dni BY CLIENTES.dni
            WHERE VENTAS.dni = CLIENTES.dni
            AND VENTAS.color = "rojo"
            AND CONCESIONARIOS.cifc = VENTAS.cifc
            AND CONCESIONARIOS.ciudad = "Madrid") = 0
```

nombre
Juan
María
Javier
Ana

♦ Problema 7.65

Obtener los nombres de los concesionarios que suministran todos los coches.

Solución:

Para resolver el problema se han utilizado dos condiciones de la forma *ANY (argumento) = 0* anidadas.

```
RETRIVE (CONCESIONARIOS.nombre)
WHERE ANY (COCHES.codcoche BY CONCESIONARIOS.cifc
    WHERE ANY (DISTRIBUCION.codcoche BY
        CONCESIONARIOS.cifc, COCHES.codcoche
        WHERE DISTRIBUCION.cifc = CONCESIONARIOS.cifc
        AND DISTRIBUCION.codcoche = COCHES.codcoche)=0)=0
```

cifc

♦ Problema 7.66

En las tuplas de la relación DISTRIBUCION que tengan una valor en el atributo *codcoche* igual a '0017', cambiar el valor del atributo *cifc* a '0002' y el valor del atributo cantidad aumentarlo en 5 unidades.

Solución:

```
REPLACE DISTRIBUCION (cifc = 0002,
                      cantidad = DISTRIBUCION.cantidad + 5)
WHERE DISTRIBUCION.codcoche  = 0017
```

♦ Problema 7.67

Duplicar todas las cantidades de la relación DISTRIBUCION que se encuentran en la misma tupla que un valor del atributo *cifc* igual a '0001'.

Solución:

```
REPLACE DISTRIBUCION (cantidad = DISTRIBUCION.cantidad * 2)
WHERE DISTRIBUCION.cifc  = 0001
```

♦ Problema 7.68

Cambiar en la base de datos todos los valores de *cifm* iguales a '0001' de tal forma que tomen el valor '0007'.

Solución:

El problema consiste en cambiar los valores de *cifm* iguales a '0001' en todas las relaciones que aparece (MARCAS y MARCO).

```
REPLACE MARCAS (cifm = 0007) WHERE MARCAS.cifc  = 0001

REPLACE MARCO (cifm = 0007) WHERE MARCO.cifc  = 0001
```

♦ Problema 7.69

Eliminar de la relación COCHES la tupla cuyo valor en el atributo *codcoche* es igual a '0020'.

Solución:

```
DELETE COCHES WHERE COCHES.codcoche  = 0020
```

♦ Problema 7.70

Eliminar todas las tuplas de la relación DISTRIBUCION para las que el atributo cantidad es mayor que 10.

Solución:

```
DELETE DISTRIBUCION WHERE DISTRIBUCION.cantidad > 10
```

♦ Problema 7.71

Añadir en la relación CLIENTES la tupla:

dni	*nombre*	*apellido*	*ciudad*
0007	Pedro	Muñoz	Càceres

Solución:

```
APPEND TO CLIENTES
        (dni = 0007,
        nombre = 'Pedro',
        Apellido = 'Muñoz',
        ciudad = 'Cáceres')
```

Capítulo 8
Formas Normales

8.1 Introducción

Supóngase una base de datos que contiene información sobre vendedores, artículos y cantidades con los siguientes datos.

codven	*ciudad*	*pais*	*codpieza*	*cantidad*
S1	Londres	Inglaterra	P1	200
S1	Londres	Inglaterra	P2	1300
S1	Londres	Inglaterra	P4	345
S2	Madrid	España	P1	24
S2	Madrid	España	P6	988
S3	París	Francia	P2	200
S4	Londres	Inglaterra	P3	34

Tabla 8.1

Es fácil darse cuenta, que esta base de datos no se encuentra demasiado bien diseñada debido al alto grado de redundancia que existe es decir; no es necesario especificar tantas veces que el vendedor S1 es de Londres ya que con que hubiese una sola referencia a este hecho, sería suficiente.

Esta redundancia a su vez puede provocar graves problemas por ejemplo, si el proveedor S1 cambia de ciudad, sería necesario actualizar al nuevo valor todas las tuplas en las cuales aparezca el proveedor S1. Otro de los problemas que se plantea en una base de datos de este tipo, es que en el momento en el que un proveedor deje de suministrar alguna pieza, automáticamente habría que borrar esa tupla de la tabla lo cual supondría la pérdida todos los datos de ese proveedor (su nombre, ciudad, etc).

Todo esto (y algunas cosas más) reafirman la idea de que esta base de datos se encuentra mal diseñada aunque aparentemente y debido al número tan pequeño de atributos presentes su diseño parece de lo más obvio.

Las *formas normales* son una serie de reglas que, de cumplirse, aseguran que el esquema diseñado tendrá un buen comportamiento en cuanto a redundancia, pérdida de información y representación de la misma.

8.2 Dependencias funcionales

Dada una relación R con atributos (X,Y,Z,...), se dice que el atributo Y de R *depende funcionalmente* del atributo X de R si y sólo si un solo valor de Y en R está asociado a cada valor de X en R. Este hecho se expresa habitualmente de la siguiente manera:

Figura 8.1: Dependencia funcional

Esto quiere decir que cada valor de X que aparezca en la relación R siempre lleva asociado el mismo valor de Y. Estas dependencias se representan también mediante los llamados diagramas de dependencias funcionales:

Figura 8.2: Diagrama de dependencias funcionales

También es imprescindible definir el concepto de *dependencia funcional completa.*

Se dice que el atributo Y de la relación R *depende funcionalmente de manera completa* del atributo X de R si depende funcionalmente de X y no depende funcionalmente de ningún otro subconjunto propio de X. Este tipo de dependencias se expresan gráficamente de la misma manera que las anteriores pero con la flecha rellena.

Figura 8.3: Dependencia funcional completa

8.3 Dependencias multivaluadas

Dada una relación R con los atributos A, B y C, la *dependencia multivaluada* (DMV)

Figura 8.4: Dependencia multivaluada

se cumple en R, si y sólo si el conjunto de valores correspondientes a un par dado (valor de A, valor de C) en R, depende sólo del valor de A y es independiente del valor de C. Los atributos A, B y C pueden ser simples o compuestos.

Se puede demostrar fácilmente que dada la relación R(A,B,C), la DMV

se cumple si y sólo si también se cumple:

Como puede verse en la figura 8.4, las dependencias multivaluadas se representan en este texto mediante una flecha terminada en una barra vertical.

8.4 Dependencias de reunión

Una *dependencia de reunión* DR es una restricción sobre una relación en cuestión. Se dice que la relación R satisface la *dependencia de reunión* (X,Y,...,Z) si y sólo si R es igual a la reunión de sus proyecciones según X,Y,...,Z donde X,Y,...,Z son subconjuntos del conjunto de atributos de R.

Las dependencias de reunión vienen dadas por restricciones que el diseñador impone sobre la tabla, tales como la siguiente:

Supóngase una relación con datos sobre vendedores, países y piezas.

vendedor	*pieza*	*país*
Pepe	Clavo	España
Pepe	Tuerca	Brasil
Juan	Clavo	Brasil
Pepe	Clavo	Brasil

en la que se impone la siguiente restricción:

Si PEPE vende CLAVOS,
Los CLAVOS se venden en BRASIL
y PEPE vende en BRASIL
Entonces PEPE vende CLAVOS en BRASIL

lo cual supone que:

Si	aparece	la pareja (Pepe,Clavo)
y		la pareja (Clavo,Brasil)
y		la pareja (Pepe,Brasil)
entonces		la tripleta (Pepe, Clavo, Brasil)

¡ debe aparecer !.

Esto hace que la relación anterior sea 3-descomponible, es decir, es igual a la reunión de las proyecciones (vendedor, pieza), (vendedor, país) y (pieza, país) y que por tanto existe la siguiente DR((vendedor, pieza), (vendedor, país) y (pieza, país)).

8.5 Primera forma normal

Una relación R se encuentra en *primera forma normal (1FN)* si y sólo si todos los dominios simples subyacentes contienen solo valores atómicos. Es decir, el cruce de fila con columna sólo tiene un dato el cual no puede descomponerse en mas.

8.6 Segunda forma normal

Una relación R se encuentra en *segunda forma normal (2FN)* si y sólo si está en 1FN y todos los atributos no clave dependen funcionalmente de manera completa de la clave primaria.

Si una relación no se encuentra en 2FN se puede aplicar el siguiente teorema de manera que, al descomponer la relación original en dos, por lo menos una de ellas se encuentre en 2FN.

- **Teorema I**

  Sea una relación R con atributos (A,B,C,D) y con clave primaria (A,B) tal que R.A⟶R.D. Entonces la relación R puede descomponerse como:

$$R \Rightarrow \begin{cases} R1(A,D) \\ R2(A,B,C) \end{cases}$$

8.7 Tercera forma normal

Una relación R se encuentra en *tercera forma normal (3FN)* si y sólo si está en 2FN y todos los atributos no clave dependen de manera no transitiva de la clave primaria.

Por tanto no están permitidas situaciones de este tipo:

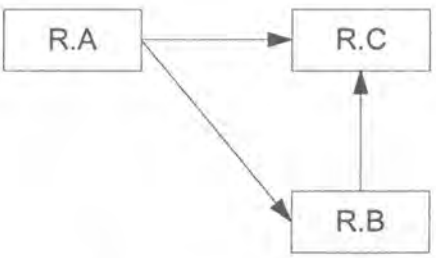


Figura 8.5: Transitividad

Si una relación no se encuentra en 3FN se puede aplicar el siguiente teorema, de manera que al descomponer la relación original en dos por lo menos una de ellas se encuentre en 3FN.

- **Teorema II**

  Sea una relación R(A,B,C) con clave primaria (A) y tal que R.B⟶R.C. Entonces la relación R puede descomponerse como:

$$R \Rightarrow \begin{cases} R1(A,B) \\ R2(B,C) \end{cases}$$

8.8 Forma normal de Boyce-Codd

Para definir la forma normal de Boyce-Codd (FNBC) es conveniente introducir primero un nuevo término. Se define un *determinante* como un atributo del cual depende funcionalmente de manera completa algún otro atributo.

Una relación está en FNBC si y sólo si todo determinante es una clave candidata.

En realidad son muy pocos los casos de relaciones que se encuentran en 3FN y no están en FNBC. Este tipo de tablas son aquellas en las que se dan las siguientes circunstancias:

- Existan varias claves candidatas
- Las claves candidatas son compuestas
- Las claves candidatas se solapan (tienen por lo menos un atributo común)

En estos casos se puede aplicar el siguiente teorema de manera que al descomponer la relación original en dos, por lo menos una de ellas se encuentre en FNBC.

- **Teorema III**

  Sea una relación R(A,B,C,D) con claves candidatas (A,B) y (B,C) y tal que R.A ⟷ R.C. Entonces la relación R puede descomponerse de cualquiera de las dos siguientes maneras:

$$R \Rightarrow \begin{cases} R1(A,C) \\ R2(B,C,D) \end{cases} \qquad R \Rightarrow \begin{cases} R1(A,C) \\ R2(A,B,D) \end{cases}$$

8.9 Cuarta Forma Normal

Una relación R se encuentra en *cuarta forma normal 4FN* si y sólo si está en FNBC y no existen dependencias multivaluadas.

En caso de tener una relación en FNBC y con dependencias multivaluadas, existe un teorema demostrado por Fagin que permite descomponer sin pérdidas la relación R, deshaciendo además las dependencias multivaluadas.

- **Teorema de Fagin**

  Sea una relación R(A,B,C) con las siguientes dependencias multivaluadas:

$$A \longrightarrow\!\!\!\!\blacktriangleright\!\!| \; B$$
$$A \longrightarrow\!\!\!\!\blacktriangleright\!\!| \; C$$

  Entonces la relación R puede descomponerse como:

$$R \Rightarrow \begin{cases} R1(A,B) \\ R2(A,C) \end{cases}$$

8.10 Quinta forma normal

Una relación R se encuentra en *quinta forma normal 5FN* si y sólo si toda dependencia de reunión en R es una consecuencia de las claves candidatas.

Esto viene a decir que una relación estará en 5FN cuando esté en 4FN y siempre que no existan restricciones impuestas por el creador.

Cuando una tabla se encuentra en 4FN y no en 5FN la solución es n-descomponerla para deshacer las DR.

8.11 Problemas Resueltos

♦ Problema 8.1

Dada la siguiente relación:

dni	*nombre*	*calle*	*ciudad*
31276123	Luis	Bravo Murillo	Madrid
52233364	Antonio	Bravo Murillo	Barcelona
1291621	Luis	Goya	Sevilla

normalizarla (si no lo está) hasta 5FN.

Solución:

Lo primero que hay que hacer siempre que se va a normalizar una relación es trazar un diagrama de dependencias funcionales de la misma. Para ello lo mejor es localizar las claves de la relación e ir viendo que, atributos dependen de ellas. Después se intentará ver si existe alguna dependencia entre las claves y por último entre el resto de los atributos.

En este caso, el diagrama es extremadamente sencillo, tal y como aparece a continuación.

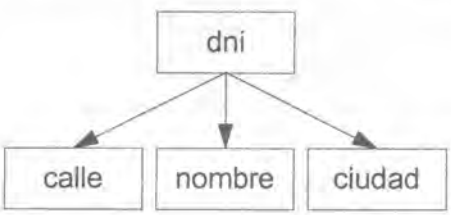


Figura 8.6: Solución problema 1

La única clave existente es el *dni* que por el hecho de serlo, proporciona dependencia funcional completa al resto de los atributos. Además entre ellos no existe ningún tipo de dependencia debido a que pueden existir las mismas calles en ciudades distintas y a que los nombres pueden aparecer repetidos aunque cada uno lleve un *dni* distinto. Con todo esto se pueden hacer las siguientes afirmaciones:

- La relación se encuentra en 1FN debido a que los dominios son atómicos, es decir sólo se tiene un dato en cada casilla.
- La relación se encuentra en 2FN debido a que todas las dependencias funcionales son completas.
- La relación se encuentra en 3FN debido a que no existen dependencias funcionales transitivas.
- La relación se encuentra en FNBC ya que está en 3FN y no se dan las tres características en cuanto a solapamiento de claves.
- La relación se encuentra en 4FN debido a que no existe ninguna dependencia multivaluada.
- Por último afirmar que también se encuentra en 5FN porque no existe ninguna dependencia de reunión.

Por tanto, la relación original se encuentra en la mejor situación posible.

♦ Problema 8.2

Dada la siguiente relación, utilizada para almacenar información sobre los artículos que un dependiente vende, además de información del propio dependiente, normalizarla (si no lo está) hasta 5FN.

dni	*calle*	*ciudad*	*codigo*	*cantidad*
31276123	Bravo Murillo	Madrid	1	10
31276123	Bravo Murillo	Madrid	2	3
52233364	Bravo Murillo	Barcelona	1	4
1291621	Goya	Sevilla	3	7

Solución:

Al igual que se dijo en el problema anterior, lo primero será hacer un diagrama de dependencias funcionales de la relación, para lo cual es útil localizar primero la clave primaria y establecer a partir de ella las dependencias existentes.

Es fácil darse cuenta que el único conjunto de atributos mediante el cual se puede distinguir una tupla de otra es el atributo compuesto (*dni*, *codigo*) y por tanto no existe ninguna otra clave candidata. El diagrama queda entonces como sigue:

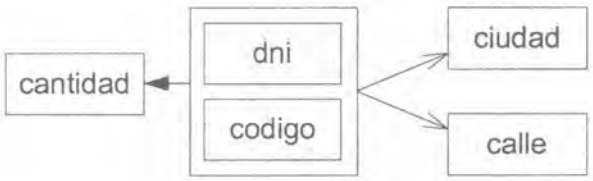


Figura 8.7: Modelo inicial

Sin embargo, es sencillo ver que las dependencias de este diagrama no son completas, ya que los atributos *calle* y *ciudad* dependen también del atributo *dni* por sí solo.

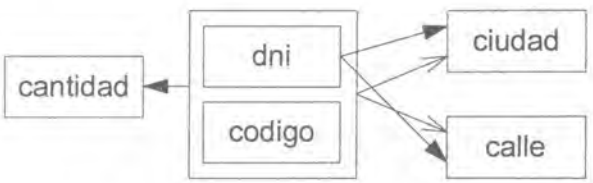


Figura 8.8: Solución al problema 8.2

- La relación se encuentra en 1FN debido a que los dominios son atómicos, es decir sólo se tiene un dato en cada casilla.
- La relación NO se encuentra en 2FN debido a que existen dependencias funcionales que no son completas. Para solucionar esto es necesario aplicar el Teorema I.

Según el Teorema I, se asociarán los atributos A, B, C y D como:

A=(dni)
B=(codigo)
C=(cantidad)
D=(calle, ciudad)

por ello la relación se descompone en:

$$R \Rightarrow \begin{cases} R1(A,D) \\ R2(A,B,C) \end{cases}$$

con lo que el esquema inicial queda ahora así:

R1=(dni, calle, ciudad)
R2=(dni, codigo, cantidad)

Ahora habrá estudiar los diagramas de dependencias funcionales de cada relación, y deducir si R1 y R2 se encuentran en 2FN.

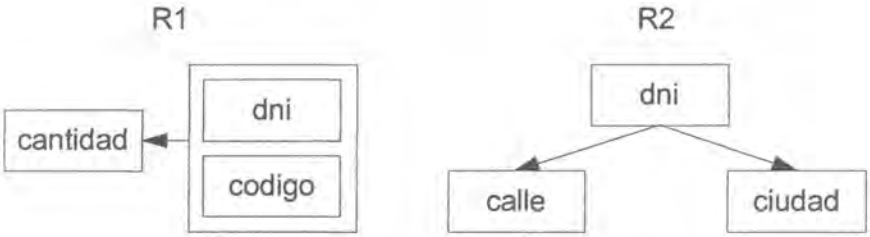


Figura 8.9: Descomposición final del problema 8.2

Como puede verse, las dos relaciones ya se encuentran en 2FN debido a que todas las dependencias funcionales son completas. Siguiendo ahora con el desarrollo:

- Las relaciones se encuentran en 3FN debido a que no existen dependencias funcionales transitivas.
- Las relaciones se encuentran en FNBC ya que está en 3FN y no se dan las tres características en cuanto a solapamiento de claves.
- Las relaciones se encuentran en 4FN debido a que no existe ninguna dependencia multivaluada.
- Por último afirmar que también se encuentran en 5FN porque no existe ninguna dependencia de reunión.

El resultado entonces de haber normalizado la relación R ha sido la división de esta en otras dos relaciones llamadas R1 y R2.

♦ Problema 8.3

Normalizar la siguiente relación:

dni	*calle*	*ciudad*	*comunidad*	*codigo*	*cantidad*
31276123	Bravo Murillo	Madrid	Madrid	1	10
31276123	Bravo Murillo	Madrid	Madrid	2	3
52233364	Bravo Murillo	Barcelona	Cataluña	1	4
1291621	Goya	Sevilla	Andalucía	3	7

Solución:

Como siempre, lo primero es realizar el diagrama de dependencias funcionales de la tabla a normalizar.

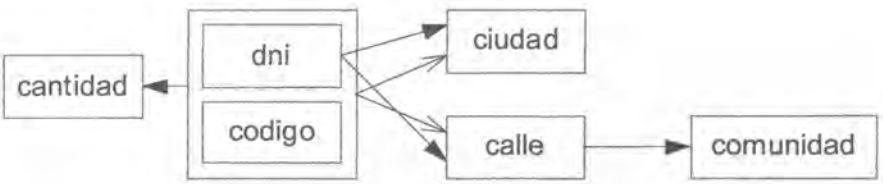


Figura 8.10: Modelo inicial

El significado de esta tabla es prácticamente igual que la del ejercicio anterior es decir, información sobre los empleados de una empresa (*dni*, *calle* y *ciudad*) además de los artículos y el número de ellos que se han vendido. La diferencia aquí es que se ha añadido el atributo *comunidad* debido al cual el conjunto de dependencias funcionales generado es algo más complicado. Teniendo en cuenta ahora el diagrama de la figura 8.10, es fácil ver que:

- La relación se encuentra en 1FN debido a que los dominios son atómicos, es decir sólo se tiene un dato en cada casilla.
- La relación NO se encuentra en 2FN debido a que existen dependencias funcionales que no son completas. Para solucionar esto es necesario aplicar el Teorema I.

Según aparece dicho teorema se asociarán los atributos A, B, C y D de la siguiente manera:

A=(dni)
B=(codigo)
C=(cantidad)
D=(calle, ciudad, comunidad)

por ello, la relación se descompone entonces en:

$$R \Rightarrow \begin{cases} R1(A, D) \\ R2(A, B, C) \end{cases}$$

con lo que el esquema inicial queda ahora así:

R1=(dni, calle, ciudad, comunidad)
R2=(dni, codigo, cantidad)

Ahora habrá que estudiar los diagramas de dependencias funcionales de las relaciónes R1 y R2, y ver si ambas se encuentran en 2FN.

Como puede verse en la figura 8.11, las dos relaciones ya se encuentran en 2FN debido a que todas las dependencias funcionales son completas. Siguiendo ahora con el desarrollo:

- La relación R2 se encuentra en 3FN debido a que no existen dependencias funcionales transitivas; sin embargo, la relación R1 NO lo está ya que como puede observarse en la figura 8.11, existe una dependencia funcional transitiva entre los atributos *dni*, *ciudad* y *comunidad*.

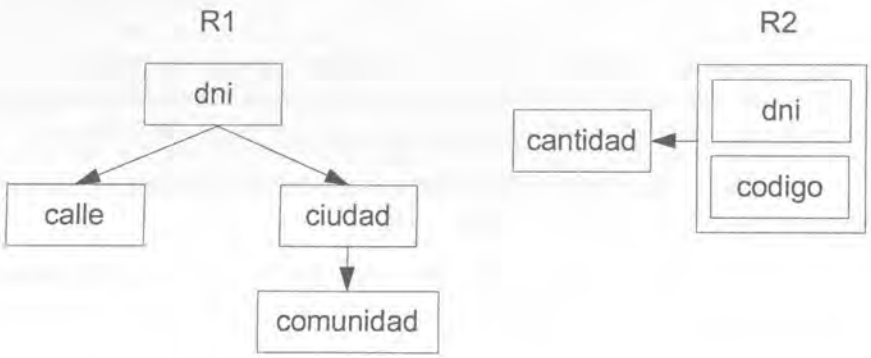


Figura 8.11: Primera descomposición

Para solucionar este problema de transitividad y conseguir que R1 cumpla 3FN, es necesario utilizar el Teorema II sobre la dependencia transitiva. Según aparece en dicho teorema, se asociarán los atributos A, B y C como:

A=(dni, calle)
B=(ciudad)
C=(comunidad)

por ello, aplicando el teorema, la relación R1 se descompone ahora en:

$$R1 \Rightarrow \begin{cases} R3(A,B) \\ R4(B,C) \end{cases}$$

con lo que el esquema inicial queda ahora así:

R3=(dni, ciudad, calle)
R2=(ciudad, comunidad)

El atributo *calle* se ha dejado asociado a la relación R3 en la cual aparece el atributo *dni*, ya que depende de él de manera completa. Los diagramas de las relaciones finales aparecen ahora en la figura 8.12.

Vistos los diagramas anteriores, se puede afirmar que las relaciones R2, R3 y R4 se encuentran en 3FN debido a que no existen dependencias funcionales transitivas. Siguiendo con el proceso:

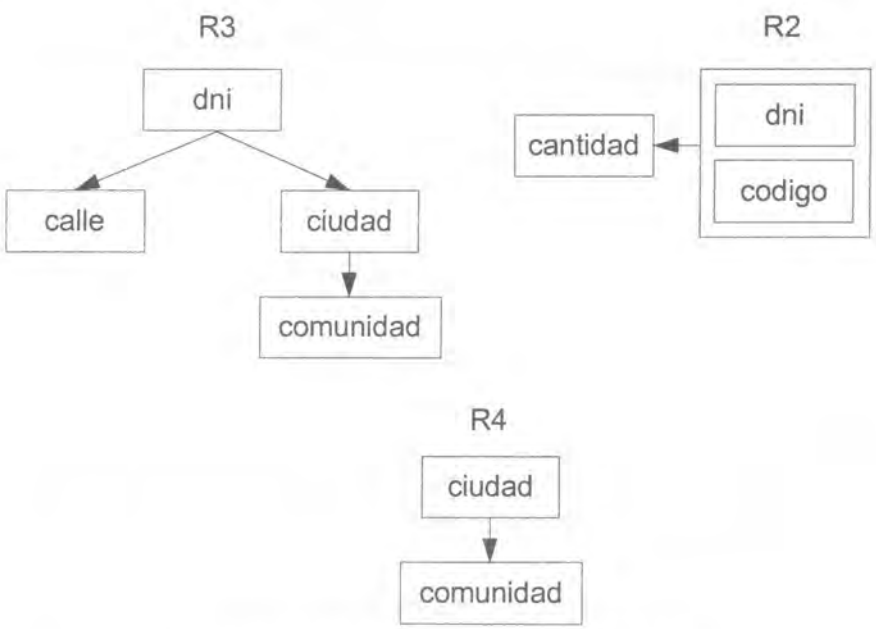


Figura 8.12: Resultado final del problema 3.

- Las relaciones R2, R3 y R4, se encuentran en FNBC ya que está en 3FN y todos los determinantes son claves candidatas.

Como puede verse en la figura 8.12, los únicos determinantes que existen en el diagrama de dependencias funcionales, son los atributos (*dni*, *codigo*), (*dni*) y (*ciudad*) los cuales funcionan como claves en las relaciones R2, R3 y R4 respectivamente.

- Las relaciones se encuentran en 4FN debido a que no existe ninguna dependencia multivaluada.
- Por último afirmar que también se encuentran en 5FN porque no existe ninguna dependencia de reunión.

El resultado de haber normalizado la relación R ha sido la división de esta en tres relaciones llamadas R2, R3 y R4.

♦ Problema 8.4

Normalizar hasta 5FN una relación R(a,b,c,d) cuyo diagrama de dependencias funcionales es el siguiente:

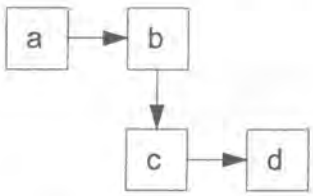


Figura 8.13: Diagrama inicial

Solución:

Dado el diagrama de dependencias funcionales de la figura 8.13, y suponiendo que los atributos que se encuentran en R son atómicos, se pueden realizar las siguientes afirmaciones:

- La relación se encuentra en 1FN debido a que los dominios son atómicos, es decir sólo se tiene un dato en cada casilla.
- La relación se encuentra en 2FN debido a que todas las dependencias funcionales son completas.

- La relación NO se encuentra en 3FN debido a que existen dos dependencias transitivas tal y como puede apreciarse en la siguiente figura:

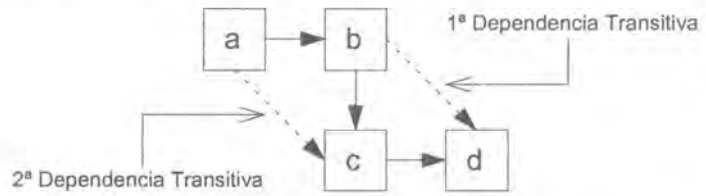


Figura 8.14: Localización de las dependencias transitivas

Como puede observarse en la figura 8.14, existen dos dependencias transitivas que hacen que la relación R no se encuentre en 3FN, por lo que será necesario aplicar el Teorema II para dividir esta tabla en dos o más y deshacer así las dependencias no deseadas. El problema surge ahora en cómo utilizar el teorema en una relación de este tipo en la que existen más de una dependencia transitiva encadenada. La solución consiste en aplicarlo de atrás hacia delante sucesivas veces es decir, deshacer primero la primera dependencia transitiva para luego deshacer la segunda.

Aplicando entonces por primera vez el Teorema II y eligiendo los atributos como:

A=(a, b)
B=(c)
C=(d)

la relación R se descompone entonces en :

$$R \Rightarrow \begin{cases} R1(A,B) \\ R2(B,C) \end{cases}$$

con lo que el esquema inicial queda como sigue:

R1=(a, b, c)
R2=(c, d)

Si se dibujan ahora los diagramas de dependencias funcionales de las relaciones R1 y R2:

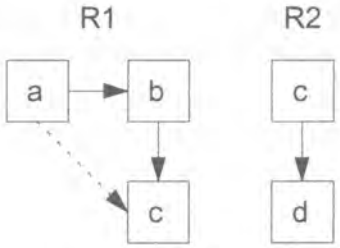


Figura 8.15: Primera descomposición

Según el diagrama de la figura. 8.15, se puede comprobar que mientras la relación R2 se encuentra ya en 3FN, la relación R1 todavía contiene una dependencia transitiva que será necesario deshacer aplicando otra vez el Teorema II. Para ello se eligen ahora los atributos como:

A=(a)
B=(b)
C=(c)

según lo cual, la relación R1 se descompone ahora en :

$$R1 \Rightarrow \begin{cases} R3(A,B) \\ R4(B,C) \end{cases}$$

con lo que el esquema inicial de la relación R1 queda ahora descompuesto de la forma siguiente:

R3=(a, b)
R4=(b, c)

Si se dibujan los diagramas de dependencias funcionales de las relaciones R2, R3 y R4:

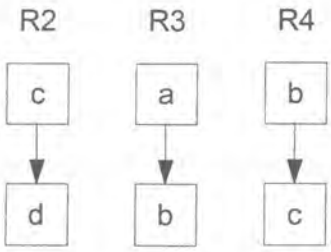


Figura 8.16: Solución final

Ahora, tal y como aparece en el diagrama anterior, las tres relaciones se encuentran ya en 3FN. Siguiendo entonces con el desarrollo:

- Las relaciones R2, R3 y R4 se encuentra en FNBC, ya que está en 3FN y todos los determinantes son claves candidatas.
- Las relaciones R2, R3 y R4 se encuentra en 4FN debido a que no existe ninguna dependencia multivaluada.
- Por último afirmar que también se encuentran en 5FN porque no existe ninguna dependencia de reunión.

El resultado de haber normalizado la relación R ha sido la división de esta en tres relaciones llamadas R2, R3 y R4.

♦ Problema 8.5

Normalizar hasta 5FN una relación R(a,b,c,d,e) cuyo diagrama de dependencias funcionales es el siguiente:

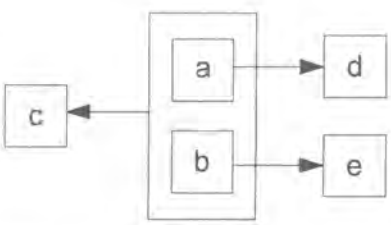


Figura 8.17: Diagrama inicial

Solución:

Dado el anterior diagrama de dependencias funcionales, y suponiendo que los dominios de los atributos son atómicos, se puede afirmar lo siguiente:

- La relación se encuentra en 1FN debido a que los dominios son atómicos, es decir sólo se tiene un dato en cada casilla.
- La relación NO se encuentra en 2FN debido a que existen dos dependencias funcionales que no son completas respecto a la clave primaria (*a*,*b*). Por tanto es necesario aplicar el Teorema I dos veces sobre la tabla, tal y como se describe a continuación.

Para aplicar el teorema por primera vez se eligen los atributos de la siguiente manera:

A=(a)
B=(b)
C=(c, e)
D=(d)

con lo que, aplicando el teorema, la relación original se descompone en:

$$R \Rightarrow \begin{cases} R1(A,D) \\ R2(A,B,C) \end{cases}$$

que al sustituir el valor de los atributos resulta:

R1=(a, d)
R2=(a, b, c, e)

Si se dibujan los diagramas de dependencias funcionales de las relaciones R1 y R2 se obtiene:

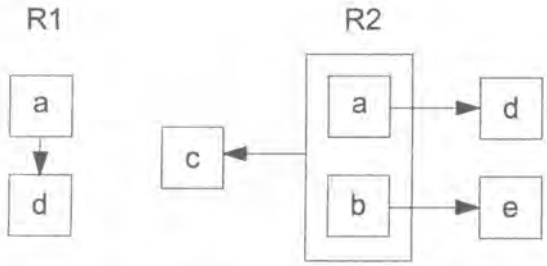


Figura 8.18: Primera descomposición

lo que significa que la relación R1 se encuentra ya en 2FN mientras que R2 todavía sigue presentando una dependencia funcional no completa, por lo que se hace necesaria otra vez la aplicación del Teorema I. Para ello se eligen ahora los atributos como sigue:

A=(b)
B=(a)
C=(c)
D=(e)

con lo que aplicando el teorema, la relación R2 se descompone en:

$$R2 \Rightarrow \begin{cases} R3(A,D) \\ R4(A,B,C) \end{cases}$$

que al sustituir el valor de los atributos resulta:

R3=(b, e)
R4=(a, b, c)

Si se dibujan ahora los diagramas de dependencias funcionales de R1, R3 y R4 resultantes, se obtiene lo representado en la figura 8.19, lo cual significa que las tres relaciones se encuentran ya en 2FN debido a que todas las dependencias funcionales son ahora completas.

Siguiendo entonces con el proceso de normalización:

- Las relaciones R1, R3 y R4 se encuentran en 3FN debido a que no existen dependencias funcionales transitivas.
- Las relaciones R1, R3 y R4 se encuentran en FNBC, ya que están en 3FN y todos los determinantes funcionan como claves. En efecto, los atributos (*a*), (*a*, *b*) y (*b*) funcionan como claves de las relaciones R1, R4 y R3 respectivamente
- La relaciones R1, R3 y R4 se encuentran en 4FN debido a que no existe ninguna dependencia multivaluada.
- Por último afirmar que también se encuentran en 5FN porque no existe ninguna dependencia de reunión.

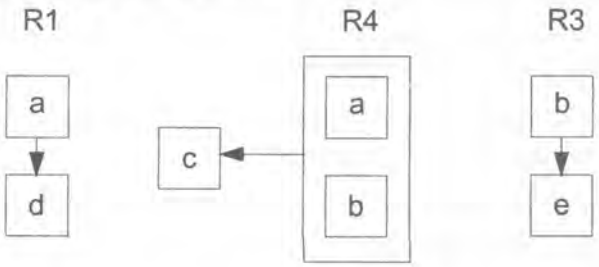


Figura 8.19: Solución final

El resultado de haber normalizado la relación R ha sido la división de esta en otras tres relaciones llamadas R1, R3 y R4.

♦ Problema 8.6

Normalizar hasta 5FN una relación R(*dni*, *nombre*, *codigo*, *cantidad*) que representa una base de datos con información sobre proveedores, códigos de piezas y cantidades que de esa pieza venden los proveedores. Se impone como condición que *NO* existen dos instancias de *nombre* repetidas en toda la relación.

Solución:

Dada la composición de esta relación, es fácil darse cuenta que existen dos posibles conjuntos de atributos que pueden funcionar como claves, ya que tanto con la pareja de atributos (*dni*, *codigo*) como con (*nombre*, *codigo*) es posible identificar de manera única cualquier tupla de la relación.

Visto este aspecto, el diagrama de dependencias funcionales de esta relación es el siguiente:

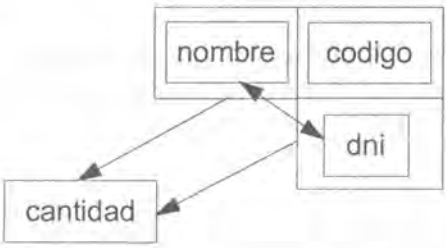


Figura 8.20: Claves solapadas

Dado el anterior diagrama de dependencias funcionales, y suponiendo que los datos están en la tabla de manera que en el cruce de fila con columna sólo hay un dato, se puede afirmar lo siguiente:

- La relación se encuentra en 1FN debido a que los dominios son atómicos, es decir sólo se tiene un dato en cada casilla.
- La relación se encuentra en 2FN debido a que todos los atributos no-clave dependen de manera completa de la clave primaria.

Esto es fácil de ver ya que el único atributo no-clave que existe es *cantidad* el cual, depende funcionalmente de manera completa tanto de la pareja de atributos (*dni*, *codigo*) como de (*nombre*, *codigo*). La única dificultad que podría plantearse para comprender esto, es el hecho de que existe una dependencia funcional biunívoca entre los atributos *nombre* y *dni*, lo cual haría suponer que en el caso de elegir como clave primaria a (dni, codigo), *nombre* no dependería funcionalmente de manera completa de ella ya que

también lo hace de *dni*; y al contrario, si se eligiese como clave primaria la pareja de atributos (*nombre*, *codigo*), *dni* no dependería funcionalmente de manera completa de ella ya que también lo hace de *nombre*. Esto no supone ningún problema para afirmar que esta relación se encuentra en 2FN ya que ninguno de estos dos atributos son no-clave debido a que forman parte de una de las dos posibles claves de la relación. Por tanto, es correcto afirmar que la relación se encuentra en 2FN debido a que todos los atributos no-clave dependen de manera completa de la clave primaria.

- La relación se encuentra en 3FN debido a que no existen dependencias funcionales transitivas, tal y como puede verse en el diagrama de dependencias funcionales de la figura. 8.20.

- La relación NO se encuentra en FNBC ya que NO todos los determinantes funcionan como claves. En efecto, los atributos *nombre* y *dni* son determinantes, ya que proporcionan dependencia funcional y sin embargo no pueden funcionar como claves de la relación

Por tanto, para llevar esta relación a FNBC, será necesario aplicar el Teorema III eligiendo los atributos de la siguiente manera:

A=(nombre)
B=(codigo)
C=(dni)
D=(cantidad)

es posible entonces descomponer sin pérdidas la relación original como sigue:

$$R \Rightarrow \begin{cases} R1(A,C) \\ R2(B,C,D) \end{cases}$$

es decir:

R1=(nombre, dni)
R2=(dni, codigo, cantidad)

La figura 8.21, muestra los diagramas de dependencias funcionales de las dos relaciones anteriores.

El resultado de aplicar el Teorema III ha sido descomponer la relación original en dos relaciones que se encuentran en FNBC.

- La relaciones R1 y R2 se encuentran en 4FN debido a que no existe ninguna dependencia multivaluada.

- Por último afirmar que también se encuentran en 5FN porque no existe ninguna dependencia de reunión.

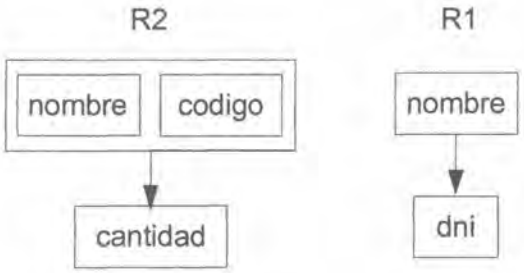


Figura 8.21: Descomposición a FNBC

En este ejemplo se ha mostrado una relación, un tanto extraña, que aun estando en 3FN no se encontraba en FNBC. Como se apuntó anteriormente, no es muy común encontrar este tipo de tablas en la realidad.

♦ Problema 8.7

Normalizar hasta 5FN una relación R(*estudiante*, *asignatura*, *profesor*) que representa una base de datos con información sobre alumnos, asignaturas y profesores que imparten las mismas, en un centro de enseñanza. Se imponen además las siguientes restricciones:

- Para cada asignatura, cada estudiante tiene sólo un profesor.
- Cada profesor sólo imparte una asignatura.
- Una asignatura puede estar dada por varios profesores.

Solución:

Como en los problemas anteriores, lo primero que hay que hacer es trazar el diagrama de dependencias funcionales de la relación en cuestión tal y como aparece en le figura 8.22. Las dependencias dibujadas en este diagrama, se pueden entender de la siguiente manera; según la primera restricción, para cada estudiante en cada materia existe un único profesor por tanto, existirá una

dependencia funcional del atributo compuesto (*estudiante*, *asignatura*) hacia el atributo (*profesor*).

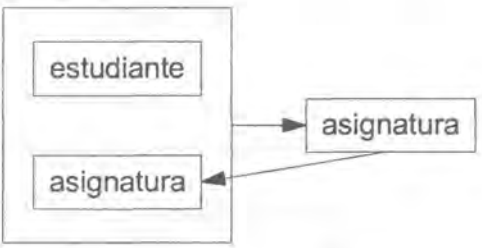


Figura 8.22: Diagrama inicial

Por otra parte y según la segunda restricción, cada profesor sólo imparte una asignatura, de manera que existirá otra dependencia funcional entre el atributo (*profesor*) y el atributo (*asignatura*).

¿Cuáles son las posibles claves candidatas de la relación?. En función de todo lo visto anteriormente es fácil darse cuenta de que existen dos posibles atributos compuestos que pueden funcionar en ese puesto; (*estudiante*, *asignatura*) y (*estudiante*, *profesor*). Pasando ya entonces al proceso de normalización:

- La relación se encuentra en 1FN debido a que los dominios son atómicos, es decir sólo se tiene un dato en cada casilla.
- La relación se encuentra en 2FN debido a que todos los atributos no clave dependen de manera completa de la clave primaria.

Este es un caso parecido al anterior en cuanto a la dependencia que existe entre los atributos *profesor* y *asignatura*, ya que este último no es un atributo no-clave.

- La relación se encuentra en 3FN debido a que no existen dependencias funcionales transitivas de la clave primaria a atributos no-clave, tal y como puede verse en el diagrama de dependencias funcionales de la figura. 8.22.
- La relación No se encuentran en FNBC ya que NO todos los determinantes funcionan como claves. En efecto, el atributo *profesor* es un determinante, ya que proporciona dependencia funcional y sin embargo no puede funcionar como clave de la relación

Otra vez, al igual que en el problema anterior se tienen dos claves candidatas solapadas, ya que cualquiera de los dos atributos compuestos (*estudiante*, *asignatura*) y (*estudiante*, *profesor*) pueden funcionar como clave.

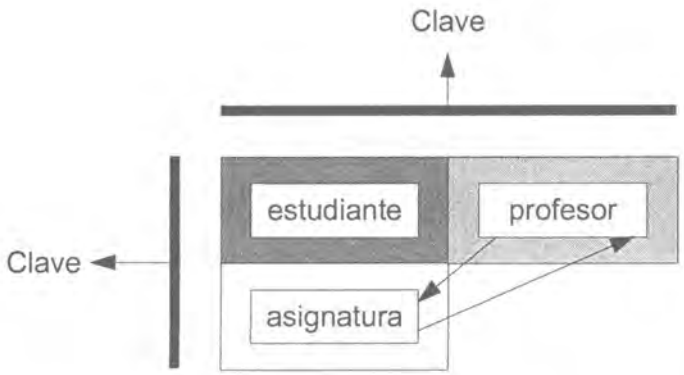


Figura 8.23: Claves Solapadas

La solución entonces será dividir la relación en dos de sus proyecciones aplicando para ello el Teorema III, para lo cual se eligen los atributos como sigue:

A=(asignatura)
B=(estudiante)
C=(profesor)
D=( )

es posible entonces descomponer sin pérdidas la relación original como sigue:

$$R \Rightarrow \begin{cases} R1(A,C) \\ R2(B,C,D) \end{cases}$$

es decir:

R1=(asignatura, profesor)
R2=(estudiante, profesor)

La relación R se ha descompuesto en dos relaciones R1 y R2 que se encuentran en FNBC.

Siguiendo con el desarrollo del problema:

- La relaciones R1 y R2 se encuentran en 4FN debido a que no existe ninguna dependencia multivaluada.

- Por último afirmar que también se encuentran en 5FN porque no existe ninguna dependencia de reunión.

En este ejemplo se ha mostrado otra relación que aun estando en 3FN no se encontraba en FNBC.

♦ Problema 8.8

Normalizar hasta 5FN una relación R(*dni*, *nif*, *ciudad*, *provincia*, *teléfono*, *codtel*) que representa una base de datos con información sobre personas físicas. El atributo *codtel* representa el prefijo telefónico de cada provincia. Se impone como restricción el que varias personas pueden tener el mismo teléfono.

Solución:

Como en los casos anteriores, el primer paso para normalizar la relación es dibujar el diagrama de dependencias funcionales y determinar las posibles claves de la tabla.

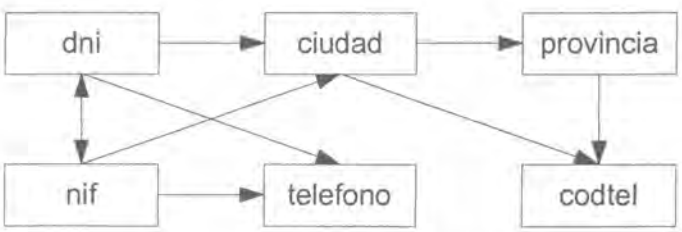


Figura 8.24: Modelo E-R inicial.

Visto el diagrama anterior y la estructura de la tabla, es fácil ver que existen dos atributos que pueden funcionar como clave candidata es decir, que mediante ellos se pueden identificar de manera única las tuplas de la relación. Estos atributos son el *dni* y el *nif*, los cuales son únicos para cada persona. Al ser los dos, posibles claves candidatas, entre ellos existe una dependencia funcional biunívoca tal y como puede verse en el diagrama anterior. De aquí en adelante se elegirá al atributo *dni* como clave primaria y a *nif* como clave candidata.

Después de este comentario preliminar, pasemos al proceso de normalización propiamente dicho.

- La relación se encuentra en 1FN debido a que los dominios son atómicos, es decir sólo se tiene un dato en cada casilla.
- La relación se encuentra en 2FN debido a que todos los atributos no clave dependen de manera completa de la clave primaria.
- La relación NO se encuentra en 3FN debido a que existen dependencias funcionales transitivas con respecto a la clave primaria.

En efecto, si se analiza el diagrama de dependencias funcionales, se ve claramente que existen dos dependencias funcionales transitivas de atributos no-clave con respecto al atributo *dni* (Fig: 8.25).

Para deshacer esto, se procederá de manera análoga al problema 8.4, es decir, aplicando dos veces el Teorema II.

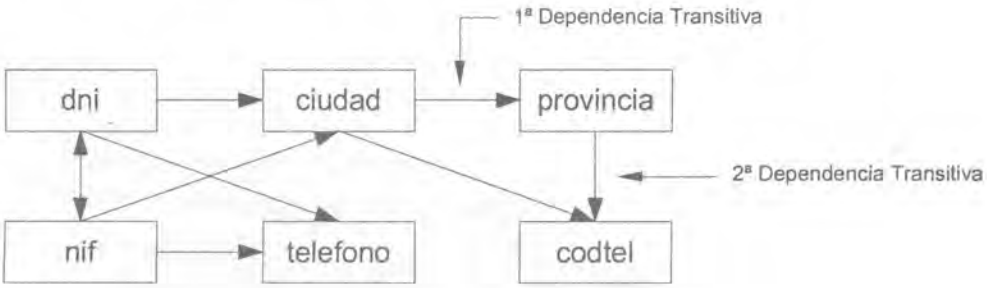


Figura 8.25: Dependencias Transitivas

En primer lugar, se eligen los atributos de la siguiente manera:

A=(dni, nif, telefono, ciudad)
B=(provincia)
C=(codtel)

produciendo la siguiente descomposición:

R1=(dni,nif,telefono,ciudad,provincia)
R2=(provincia, codtel)

Ahora será necesario aplicar otra vez el teorema sobre la relación R1, para lo cual se eligen los atributos como:

A=(dni, nif, telefono)
B=(ciudad)
C=(provincia)

resultando la siguiente descomposición:

R3=(dni, nif, telefono, ciudad)
R4=(ciudad, provincia)

Realizado todo el proceso anterior, el diagrama total de dependencias funcionales queda de la manera siguiente.

Observando los diagramas de dependencias de la figura 8.26, se puede observar que las relaciones R4 y R2 no presentan ya ningún tipo de dependencia funcional transitiva. Sin embargo, en la relación R3 aparecen todavía dos dependencias funcionales transitivas (figura 8.27) las cuales no afectan a que R3 se encuentre en 3FN ya que los atributos *nif* y *dni* son los dos claves.

Por tanto, se puede afirmar ahora que las relaciones R2, R3 y R4 se encuentran en 3FN.

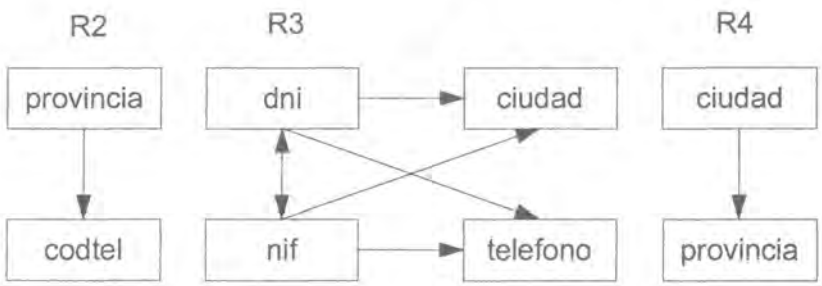


Figura 8.26: Primera descomposición

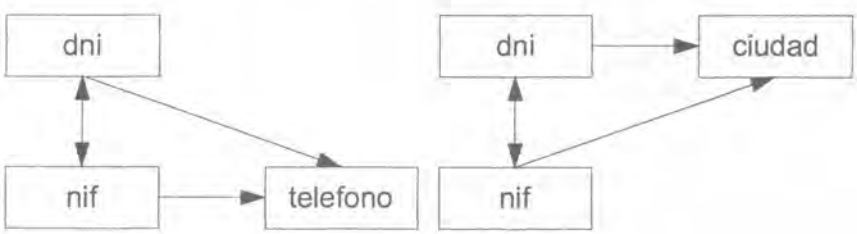


Figura 8.27: Segunda descomposición

- Las relaciones R2, R3 y R4 se encuentran en FNBC, ya que están en 3FN y todos los determinantes funcionan como claves. No existe ningún atributo determinante que no pueda funcionar como clave de las relaciones.

- La relaciones R2, R3 y R4 se encuentran en 4FN debido a que no existe ninguna dependencia multivaluada.
- Por último afirmar que también se encuentran en 5FN porque no existe ninguna dependencia de reunión.

♦ Problema 8.9

Normalizar hasta 5FN una relación R(a,b,c,d) cuyo diagrama de dependencias funcionales es el siguiente.

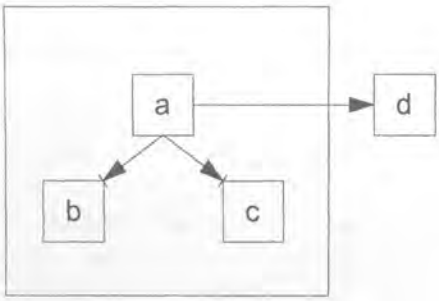


Figura 8.28: Esquema a normalizar

Solución:

Como puede observarse en el diagrama anterior, este problema presenta un tipo de dependencias que hasta ahora no se habían tratado en los ejercicios anteriores, las dependencias multivaluadas. Se verá a lo largo de la solución que el tratamiento de las mismas, aplicando los teoremas convenientes, es relativamente sencillo.

Lo primero que es necesario conocer para atacar el problema de la normalización, es la clave primaria de la relación que como puede observarse en la figura 8.28, se trata del atributo compuesto (a,b,c). Comenzando con el procedimiento habitual de normalización:

- La relación se encuentra en 1FN debido a que los dominios son atómicos es decir, se supone que sólo se tiene un dato en cada casilla.
- La relación NO se encuentra en 2FN debido a que existen dependencias funcionales que no son completas.

Ello es debido a la dependencia que aparece entre los atributos *a* y *d*, lo cual hace que este último no presente una dependencia funcional completa con respecto a la clave primaria. Por tanto, es necesario aplicar el Teorema I eligiendo los atributos de la siguiente manera:

A=(a)
B=(b, c)
C=( )
D=(d)

Aplicando entonces el Teorema I la relación se descompondrá en:

$$R \Rightarrow \begin{cases} R1(A, D) \\ R2(A, B, C) \end{cases}$$

quedando entonces descompuesta la relación inicial en:

R1=(a, d)
R2=(a, b, c)

Si se dibujan ahora los diagramas de dependencias de las relaciones resultantes R1 y R2, el resultado es:

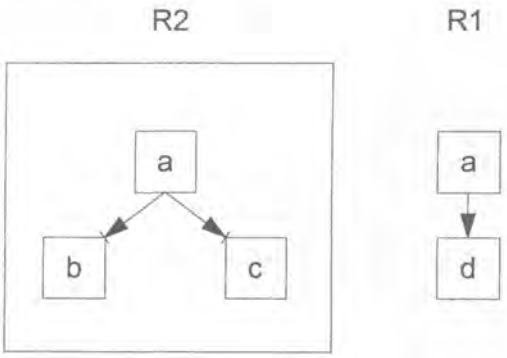


Figura 8.29: Conversión a 2FN

lo que significa que las dos relaciones ya se encuentran en 2FN. Siguiendo entonces con el proceso de normalización:

- Las relaciones R1 y R2, se encuentra en 3FN debido a que no existen dependencias funcionales transitivas de las claves primarias a atributos no-clave, tal y como puede verse en la figura 8.29.

- Las relaciones R1 y R2, se encuentran en FNBC ya que están en 3FN y todos los determinantes funcionan como claves. No existe ningún determinante que no pueda funcionar como clave de las relaciones.
- La relacion R1 se encuentran en 4FN debido a que no existe ninguna dependencia multivaluada. Sin embargo, NO se puede decir lo mismo de la relación R2 ya que presenta dos dependencias multivaluadas:

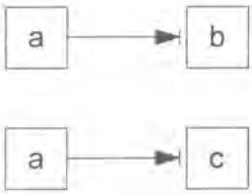


Figura 8.30: Dependencias multivaluadas.

Una relación que presenta dos dependencias multivaluadas, no se encuentra en 4FN y por tanto se hace necesario aplicar el Teorema de Fagin para dividir la relación en dos de sus proyecciones.

Según dicho teorema, y eligiendo los atributos como:

A=(a)
B=(b)
C=(c)

la relación R2 se dividirá en:

$$R2 \Rightarrow \begin{cases} R3(A,B) \\ R4(A,C) \end{cases}$$

es decir:

R3=(a, b)
R4=(a, c)

con lo que el diagrama de dependencias funcionales global quedaría ahora tal y como aparece en la figura 8.31. Como puede verse ya no existe ninguna dependencia multivaluada debido a que éstas siempre se dan en parejas. Dividida la relación R2, las relaciones R3 y R4 no presentan ahora ningún tipo de dependencia funcional ni multivaluada por lo que se puede afirmar que se encuentran ya en 4FN.

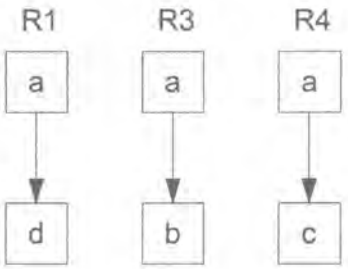


Figura 8.31: Descomposición final

- Por último afirmar que también se encuentran en 5FN porque no existe ninguna dependencia de reunión.

♦ PROBLEMA 8.10

Normalizar hasta 5FN, una base de datos de una academia que contenga información sobre cursos, profesores, libros, editorial de los libros, ciudad de la Editorial, teléfono de los profesores y aulas. Se imponen ahora las siguientes (aunque un tanto extrañas) condiciones:

- Cada Curso, es impartido siempre por un grupo bien definido de profesores.
- Cada Curso, tiene un grupo bien definido de libros (se utilizan todos ellos).
- Cada Curso impartido por un profesor con un cierto libro, se realizará en un aula distinta.

Para plasmar todas estas condiciones, se incorpora a continuación una hipotética tabla con los tipos de datos anteriormente definidos.

curso	*profesor*	*libro*	*aula*	*editorial*	*ciudad*	*teléfono*
Física	Luis	A	1	Ciencia	Madrid	212121
Física	Luis	B	2	Saber	Sevilla	212121
Física	Paco	A	3	Ciencia	Madrid	434343
Física	Paco	B	4	Saber	Sevilla	434343
Lengua	Pepe	C	5	Saber	Sevilla	545454
Lengua	Pepe	D	6	Futuro	Barcelona	545454
Lengua	Ana	C	7	Saber	Sevilla	323232
Lengua	Ana	D	8	Futuro	Barcelona	323232
Lengua	Juan	C	9	Saber	Sevilla	121212
Lengua	Juan	D	10	Futuro	Barcelona	121212

Solución:

Teniendo entonces en cuenta el enunciado del problema y los datos de la tabla anterior, se puede trazar el siguiente diagrama de dependencias funcionales:

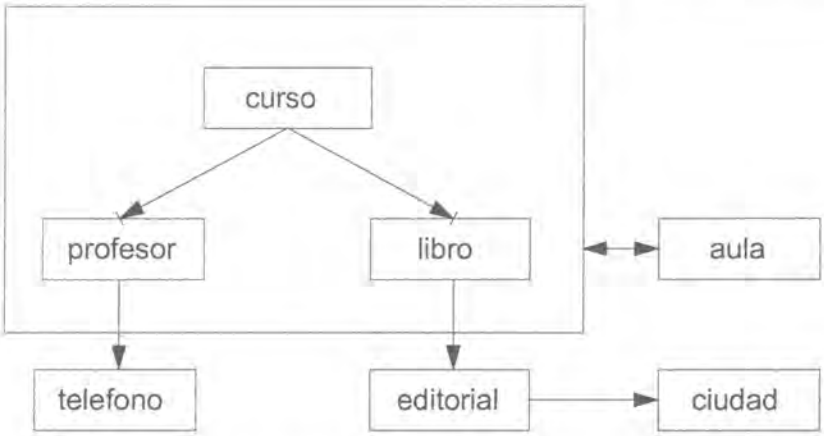


Figura8.32: Diagrama de dependencias inicial

Como puede observarse del diagrama anterior, en esta tabla se dan a la vez algunas de las situaciones vistas anteriormente es decir, dependencias no completas, dependencias transitivas y dependencias multivaluadas. Como siempre, el primer paso para comenzar el proceso de normalización es determinar cuáles son las claves de la relación y cuál de ellas funcionará como clave primaria. En este caso, es fácil ver que cualquier tupla está definida de manera única por los valores del atributo compuesto (*curso*, *profesor*, *libro*) siendo por tanto éste, el que funcionará como clave de la relación. Comenzando entonces con el proceso de normalización:

- La relación se encuentra en 1FN debido a que los dominios son atómicos.
- La relación NO se encuentra en 2FN debido a que existen dependencias funcionales no completas de la clave primaria.

Para resolver esto, como en casos anteriores, es necesario aplicar el Teorema I y dividir la tabla en dos de sus proyecciones. El problema es que en esta relación existen dos dependencias funcionales no completas, por lo que se hará necesaria la aplicación del teorema dos veces (figura 8.33).

En primer lugar, para deshacer la dependencia no completa 1 se eligen los atributos de la siguiente manera:

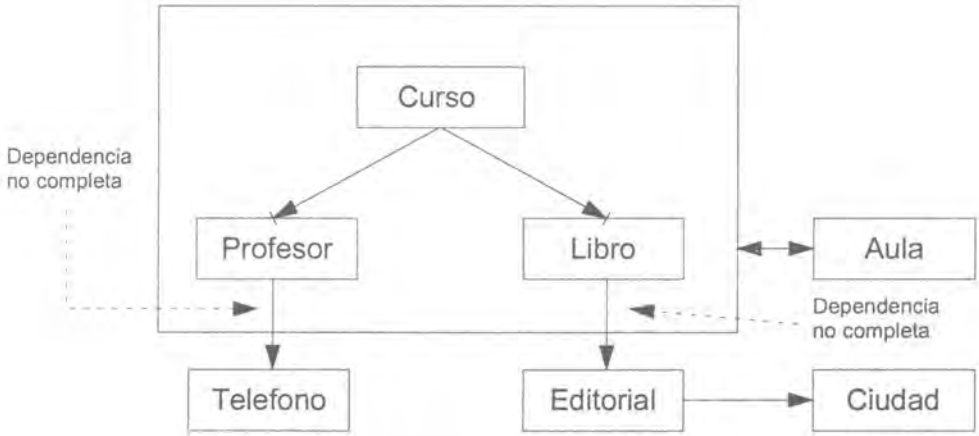


Figura 8.33: Dos dependencias no completas

A=(profesor)
B=(curso, libro)
C=(editorial, ciudad, aula)
D=(telefono)

de manera que la relación original se descompondrá en:

$$R \Rightarrow \begin{cases} R1(\text{profesor, telefono}) \\ R2(\text{profesor, curso, libro, editorial, ciudad, aula}) \end{cases}$$

quedando entonces el diagrama de dependencias funcionales como en la figura 8.34, en la que se ve que la relación R2 resultante de la primera descomposición, presenta todavía la dependencia funcional no completa llamada anteriormente *dependencia no completa 2*. Para llevar entonces a la relación R2 a 2FN será necesario entonces aplicar el Teorema I sobre dicha dependencia funcional no completa, eligiendo para ello los atributos como aparece en la figura 8.34.

A=(libro)
B=(curso, profesor)
C=(aula)
D=(editorial, ciudad)

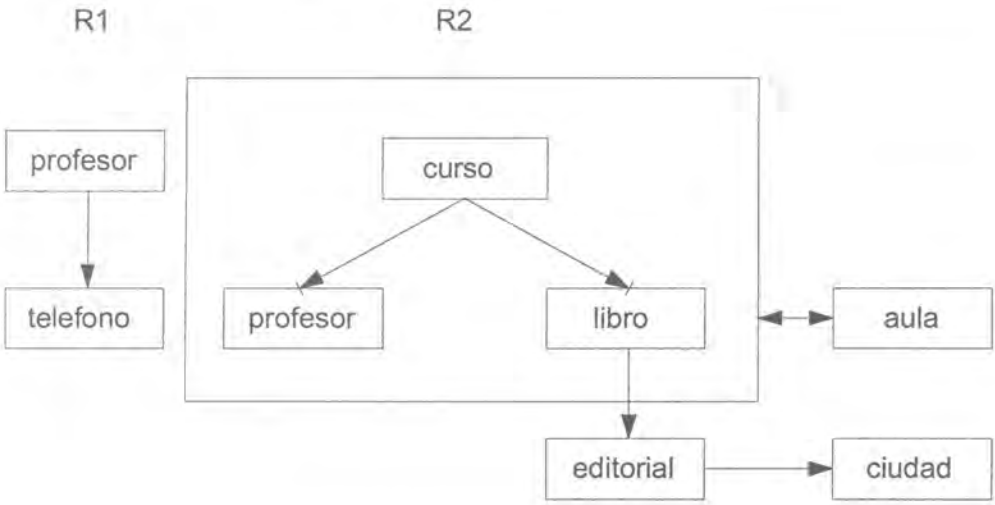


Figura 8.34: Primera descomposición

Aplicando entonces el Teorema I, la relación R2 se descompondrá en:

$$R \Rightarrow \begin{cases} R3(\text{libro}, \text{editorial}, \text{ciudad}) \\ R4(\text{libro}, \text{curso}, \text{profesor}, \text{aula}) \end{cases}$$

quedando entonces el diagrama de dependencias funcionales global tal y como aparece el la figura 8.35.

Al haber deshecho todas las dependencias funcionales no completas, ahora se puede afirmar que las relaciones R1, R3 y R4 se encuentran en 2FN.

- Las relaciones R1 y R4 se encuentra en 3FN debido a que no existen dependencias funcionales transitivas de las claves primarias a atributos no-clave, tal y como se muestra en el diagrama anterior. Sin embargo, la relación R3 NO se encuentra en 3FN ya que existe una dependencia funcional transitiva de su clave.

Para conseguir entonces que R3 se encuentre en 3FN es necesario dividirla en dos de sus proyecciones mediante el Teorema II. Eligiendo entonces los atributos como:

A=(libro)
B=(editorial)
C=(ciudad)

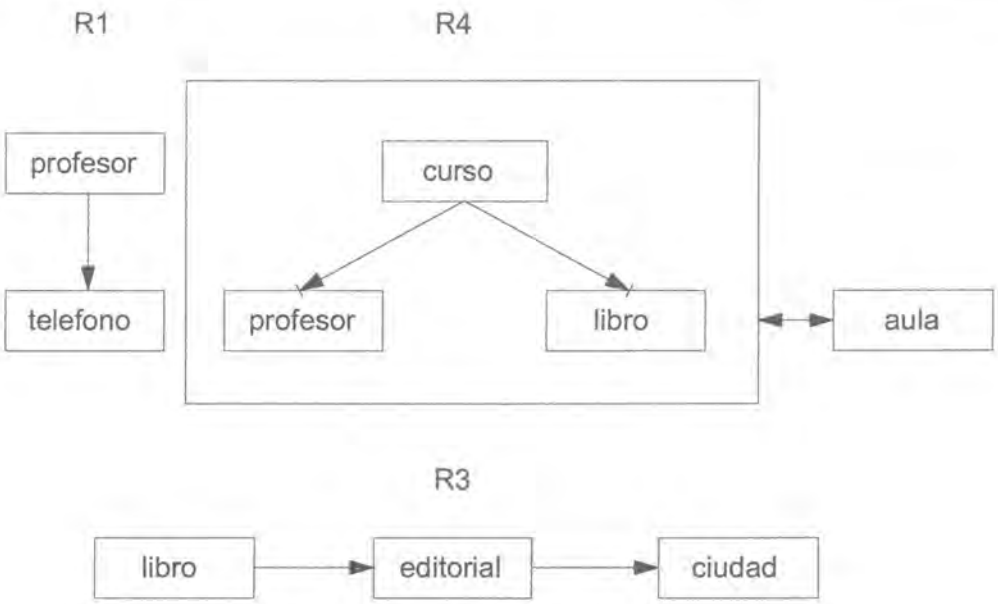


Figura 8.35: Relaciones en 2FN

$$R3 \Rightarrow \begin{cases} R5(\text{libro}, \text{editorial}) \\ R6(\text{editorial}, \text{ciudad}) \end{cases}$$

la relación R3 se descompondrá en:

con lo que el diagrama de dependencias funcionales quedará tal y como aparece en la figura 8.36.

Ahora ya se puede afirmar que las relaciones R1, R4, R5 y R6 se encuentran en 3FN.

- Las relaciones R1, R4, R5 y R6 se encuentran en FNBC ya que todos los determinantes pueden funcionar como clave de cada relación respectivamente. Notar que *curso* no es un determinante, ya que proporciona dependencias multivaluadas y no dependencias funcionales.
- Las relaciones R1, R5 y R6 se encuentran en 4FN ya que no existe en ellas ninguna dependencia multivaluada. No es este el caso de la relación R4, sobre la cual se puede decir que NO se encuentra en 4FN debido a que presenta dos dependencias multivaluadas.

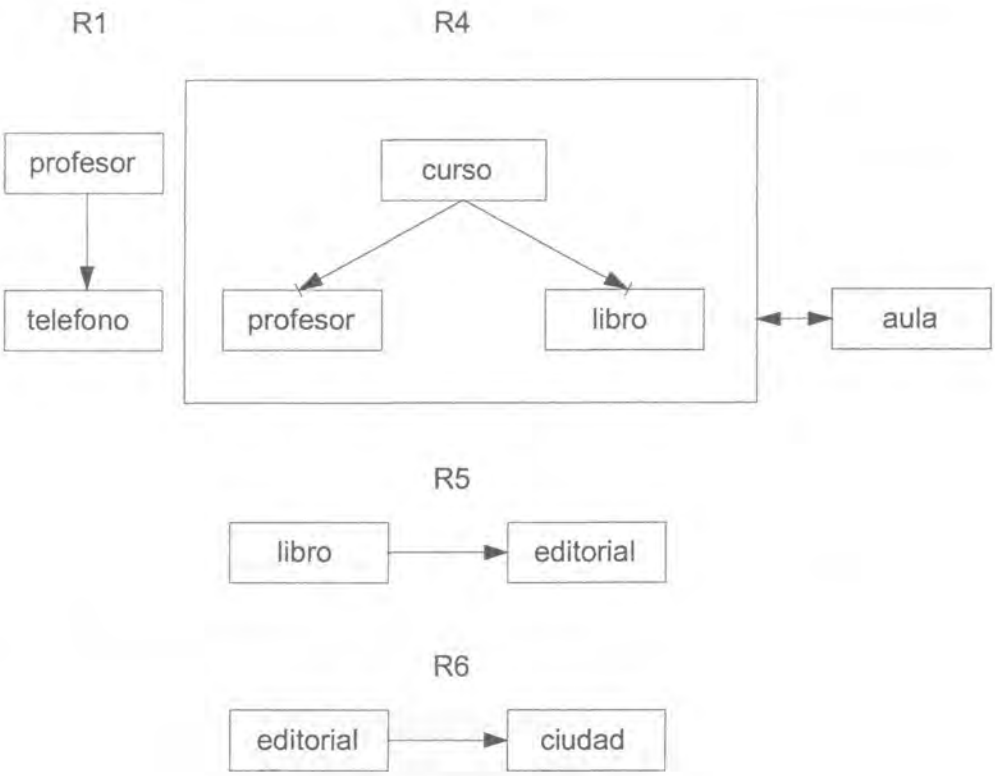


Figura 8.36: Relaciones en 3FN.

El problema que se tiene ahora enfrente es algo más complicado de lo que parece, por lo cual merece la pena analizarlo con detenimiento. Vamos a distinguir para ello tres situaciones posibles.

Supongamos primeramente que no existe el atributo *aula*. En este caso el diagrama de dependencias funcionales que presentaría la relación R4 sería el mostrado en la figura 8.37.

Por tanto, la solución para deshacer las dependencias multivaluadas es aplicar de manera directa el Teorema de Fagin, es decir, la relación R4 se descompondrá en:

$$R4 \Rightarrow \begin{cases} R7(\text{curso}, \text{profesor}) \\ R8(\text{curso}, \text{libro}) \end{cases}$$

R4

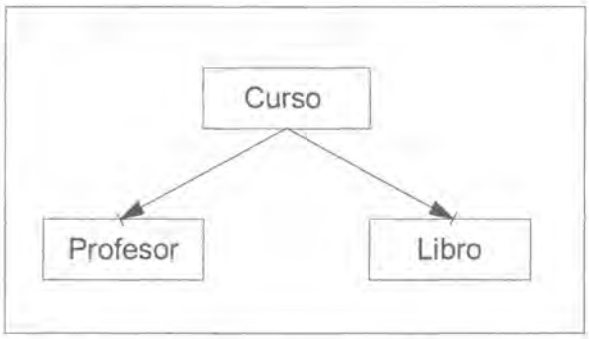


Figura 8.37: Relación R4.

obteniendo ahora dos relaciones R7 y R8 que se encuentran en 4FN y que de alguna manera han reducido la redundancia de datos existente en R4.

Vayamos ahora al problema real es decir, existe el atributo *Aula* y además éste puede funcionar como clave de la relación. La solución ahora consiste en aplicar de la misma manera el teorema de Fagin pero haciendo una "pequeña trampa" es decir, eligiendo los atributos de la siguiente manera:

A=(Curso, Aula)
B=(Profesor)
C=(Libro)

es decir, se ha añadido el atributo *aula*, que también puede funcionar como clave, al atributo A que es el que proporciona la dependencia multivaluada. Entonces, la relación R4 quedaría descompuesta en:

quedando entonces el diagrama de dependencias funcionales tal y como se

$$R4 \Rightarrow \begin{cases} R7(\text{Curso}, \text{Profesor}, \text{Aula}) \\ R8(\text{Curso}, \text{Libro}, \text{Aula}) \end{cases}$$

muestra en la figura 8.38. Ahora se puede afirmar que las relaciones R1, R5, R6, R7 y R8 se encuentran en 4FN.

No obstante es interesante comentar que en este caso, el haber pasado la relación R4 a 4FN no ha supuesto mejoras notables en cuanto a redundancia de información se refiere, ya que sigue de alguna manera existiendo cierto nivel de repetición de información.

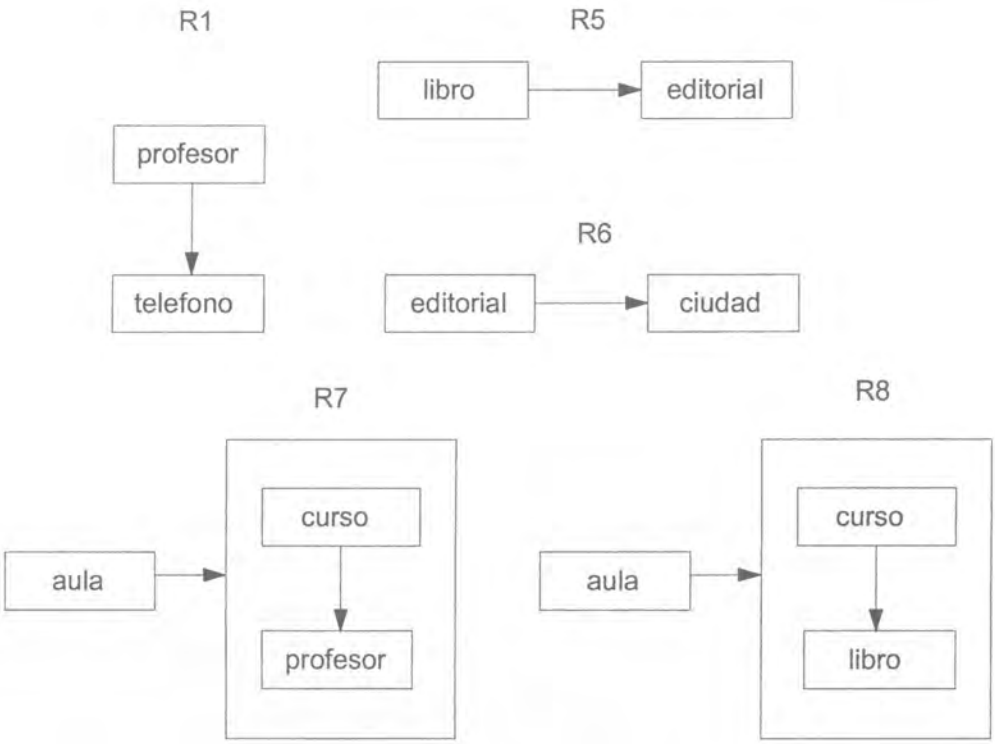


Figura 8.38: Aspecto final después de normalizar

El último caso a comentar es aquel en el que el atributo *aula* no hubiese podido funcionar como clave de la relación, es decir:

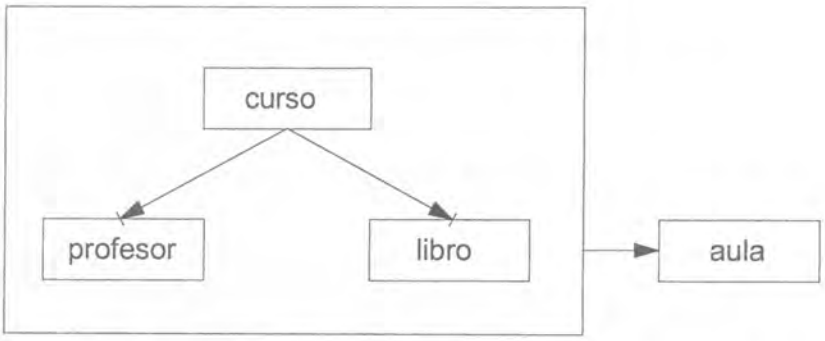


Figura 8.39: Diagrama en el que *aula* no es clave.

En este caso también es posible realizar la misma descomposición, aunque ahora sí se disminuye la redundancia. Supongamos como se ha dicho, que el atributo *aula* no es clave de la relación R4, con lo que dicha tabla podría presentar un aspecto como el siguiente.

curso	*profesor*	*libro*	*aula*
Física	Luis	A	1
Física	Luis	B	2
Física	Paco	A	2
Física	Paco	B	2
Lengua	Pepe	C	1
Lengua	Pepe	D	2
Lengua	Ana	C	1
Lengua	Ana	D	1
Lengua	Juan	C	2
Lengua	Juan	D	1

Si en este caso se realiza una descomposición como la anterior, es decir:

Es fácil ver que se cumple que $R7 \times R8 = R4$.

Ahora, centrándonos en el segundo caso que es el que concierne al problema, se puede afirmar que las relaciones R1, R5, R6, R7 y R8 se encuentran en 4FN.

- Por último afirmar que también se encuentran en 5FN porque no existe ninguna dependencia de reunión.

Capítulo 9
Modelo en Red

9.1 Introducción

El modelo en red, es actualmente uno de las tres formas de organizar la información, más utilizada en los sistemas comerciales de bases de datos. Históricamente, las estructuras y el lenguaje del modelo, fueron definidas por el comité CODASYL por lo que en mucha de la bibliografía existente, al modelo en red se le denomina modelo CODASYL. Posteriormente el ANSI (American National Standards Institute) realizó algunas modificaciones y recomendaciones sobre el lenguaje.

Como se verá a lo largo del desarrollo de este capítulo, el lenguaje de un sistema en red no es como SQL, un lenguaje autónomo en sí, sino que se compone de una serie de funciones que habrá que utilizar de manera inmersa dentro de un lenguaje anfitrión. Para este propósito y debido a que es uno de los lenguajes más difundidos, se utilizará PASCAL como lenguaje *host* aunque, en función del producto comercial en cuestión, es posible que en cada caso particular el lenguaje anfitrión sea distinto (C, COBOL etc.).

El nombre de las funciones propias de un sistema en red también varía en función del producto comercial con el que se trabaje, aunque la función en sí es la misma. Por ello en este capítulo, se nombrarán estas funciones de una manera general y sin particularizar en un producto en concreto, pero teniendo bien claro que cualquiera de las funciones utilizadas aquí tendrá una homóloga en el sistema comercial utilizado.

Visto todo esto, la idea que debe sacarse en claro es que lo importante de este capítulo es asimilar el funcionamiento y la manera de trabajar de un sistema en red.

Antes de pasar a describir y utilizar el lenguaje en sí, se definirán primero de manera muy breve las estructuras básicas de datos que componen un sistema en red.

9.2 Registros, tipos de registros y datos

En un sistema en red, un *registro* es un conjunto de valores de datos relacionados. Estos registros se clasifican en *tipos de registros*, donde cada uno de ellos describe la estructura de un grupo de registros que guardan la misma información.

Por ejemplo, un Tipo de Registro ESTUDIANTE

ESTUDIANTE			
dni	nombre	apellidos	dirección

Figura 9.1: Tipo de registro.

Mientras que una instancia de este tipo de registro sería la siguiente:

8121871	Antonio	García	Castellana 2

Figura 9.2: Instancia de registro.

A diferencia de las bases de datos relacionales, cuyos registros se engloban dentro de la estructura de la tabla relacional, los sistemas en red presentan un nuevo tipo de estructura denominado conjunto, el cual como posteriormente se verá permite enlazar unos registros con otros.

9.3 Conjuntos, Tipos de Conjuntos e Instancias

Un *tipo de conjunto* es una descripción de 1:n relaciones entre dos tipos de registros. Este tipo de estructuras se representan habitualmente por un *diagrama de Bachman*, el cual se compone generalmente de:

- Un Nombre para el tipo de conjunto
- Un tipo de registro propietario
- Un tipo de registro miembro

Un ejemplo de un conjunto que representa la relación entre las asignaturas en las cuales está matriculado un alumno podría representarse de la siguiente manera:

Figura 9.3: Tipo de conjunto.

donde el tipo de registro ESTUDIANTE es el registro propietario, ASIGNATURA es el tipo de registro miembro y ESTUDIA el tipo de conjunto.

Los tipos de conjuntos, al igual que los tipos de registros, presentan también *instancias de conjuntos* o, como aparece en gran parte de la bibliografía, *ocurrencias de conjuntos*.

Una *ocurrencia de conjunto* está compuesta por:

- Una instancia de registro propietario.
- Un número cualquiera de ocurrencias de registros miembro.
- Una estructura de datos generalmente del tipo de lista doblemente enlazada.

Una ocurrencia de conjunto podría ser por ejemplo la representada en la figura 9.4 en la que como puede verse, aparece un registro propietario (ESTUDIANTE) enlazado con tres registros miembros.

Este tipo de estructura de datos suele implementarse mediante listas enlazadas o doblemente enlazadas, de manera que desde el registro propietario es posible acceder a cualquiera de las instancias de registro que pertenecen a esa ocurrencia de conjunto (figura 9.4).

Una base de datos planteada según este modelo, consistirá entonces en una serie de tipos de conjuntos entrelazados entre sí de manera que un tipo de registro que en una ocurrencia de conjunto funciona como propietario, en otra ocurrencia de otro tipo de conjunto distinto funciona como miembro.

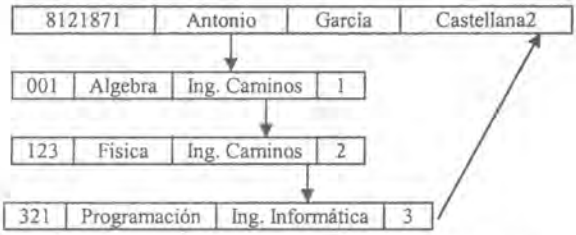


Figura 9.4: Ejemplo de ocurrencia de conjunto.

9.4 Restricciones de los miembros de un conjunto

En el modelo en red, existen ciertas restricciones que pueden especificarse sobre los miembros de un conjunto. Estas restricciones pueden ser de dos tipos:

- Restricciones de inserción.
- Restricciones de mantenimiento.
- *Las restricciones de inserción* especifican de qué manera se deben insertar los miembros de un conjunto. A diferencia de otro tipo de bases de datos, en un sistema en red cuando se definen los conjuntos es posible especificar cómo se desea que los nuevos registros se añadan a dicho conjunto (ya se verá de qué manera). Existen dos opciones:
  - *Automática*. La ocurrencia de registro que se añade a la base de datos, queda conectado automáticamente a una ocurrencia de conjunto (ya se estudiarán los comandos que hacen esto posible).
  - *Manual*. Cuando se añade una ocurrencia nueva de registro a la base de datos, ésta no se conecta directamente a ninguna ocurrencia de conjunto, sino que mediante ciertas funciones que posteriormente se verán el usuario lo debe conectar de manera manual.
- *Las restricciones de mantenimiento* especifican cómo deben existir los registros miembros dentro de una base de datos, es decir, si pueden existir sin un propietario o si por el contrario no tienen cabida en la base de datos si no están ligados a una ocurrencia de conjunto. Existen entonces tres opciones:
  - *Opcional*. Un registro miembro puede existir en la base de datos sin estar conectado a ninguna ocurrencia de conjunto. Este tipo de registros pueden conectarse y desconectarse de una ocurrencia de conjunto a voluntad.

- *Obligatoria*. Un registro miembro no puede existir en la base de datos sin su propietario, es decir, debe ser siempre miembro de alguna ocurrencia de conjunto. Puede ser reconectado de una ocurrencia de conjunto a otra en un solo paso, utilizando para ello el comando RECONNECT (que posteriormente se definirá).
- *Fija*. Un registro miembro no puede existir sin su propietario y además, una vez conectado por primera vez, no puede reconectarse a ninguna otra ocurrencia de conjunto.

9.5 Tipos especiales de conjuntos

Existen tres tipos especiales de conjuntos en el modelo CODASYL:

- *Conjuntos sistema-propietario* son aquellos en los que no existe un tipo de registro propietario como tal, sino que es el propio *sistema* el que actúa como propietario del mismo.

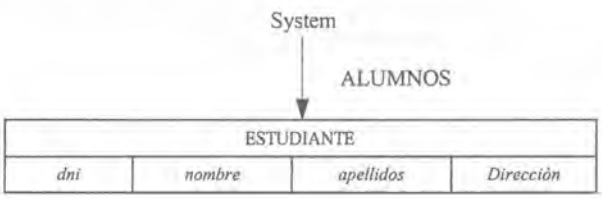


Figura 9.5: Conjunto Sistema – Propietario.

Este tipo de conjuntos tiene dos funciones muy importantes dentro de un sistema en red:

- Proporcionan un punto de entrada a la base de datos mediante la especificación de alguno de sus registros miembros.
- Pueden ser utilizados para ordenar los registros de un cierto tipo de registros usando las debidas condiciones de ordenación (que posteriormente se estudiarán).

- *Conjuntos Multimiembros* son aquellos en los que los tipos de registros miembro pueden ser más de uno.

- *Conjuntos Recursivos* son aquellos en los que el mismo tipo de registro representa el papel de miembro y propietario a la vez, como en el siguiente caso:

Figura 9.6: Conjunto recursivo.

9.6 Area de trabajo de programa

Todo programa que se ejecuta en un sistema, consta de una secuencia de sentencias; algunas son propias del lenguaje anfitrión utilizado (que en nuestro caso será PASCAL), mientras que otras serán sentencias de órdenes DBTG. Un programa de este tipo se denomina una *unidad de ejecución*. Estas sentencias acceden y manipulan tanto los elementos de la base de datos como las variables declaradas localmente.

Para cada uno de estos programas de aplicación, el sistema mantiene un *área de trabajo de programa* (llamado *área de trabajo de usuario*) en el modelo DBTG, que es un área de almacenamiento de registros intermedios que contiene las siguientes variables:

- *Plantillas de registro* que es tipo de registro definido por cada tipo de registro existente en la base de datos, y cuya función es poder cargar los datos de la misma.
- *Punteros de actualidad* que son una serie de punteros que marcan a ciertos registros accedidos en la base de datos y cuya utilidad es la de poder navegar por el sistema de conjuntos que componen la base de datos. Entre ellos están:
  - *Actual de tipo de registro*. Es un puntero que, para cada tipo de registro definido en la base de datos, indica la última ocurrencia de cada uno de ellos a la que se ha accedido.
  - *Actual tipo de conjunto*. Es un puntero que, para cada tipo de conjunto definido en el esquema de la base de datos, especifica la última ocurrencia a la que se ha accedido. Como cuando se accede a una ocurrencia de conjunto a lo que se accede de verdad es a un registro, este puntero señalará al registro accedido de esa ocurrencia de conjunto.
  - *Actual de unidad de ejecución (CRU)*. Es un solo puntero que contiene la dirección del registro más recientemente accedido por el programa de aplicación.

- *Indicadores de estado* que son un conjunto de variables que utiliza el sistema para comunicar al programa de aplicación el resultado de la última operación aplicada a la base de datos, es decir, si se ha encontrado el dato buscado o si por el contrario la búsqueda ha fallado. Entre ellos:
  - DB-STATUS. Después de cada comando DML, el valor de esta variable indica si éste finalizó con éxito o si por el contrario se produjo un error o excepción. La excepción más habitual es la de *FIN-DE-CONJUNTO* (EOS). No se trata de ningún error, ya que lo único que indica es que no existen más registros miembro en esta ocurrencia de conjunto. Se utiliza normalmente para finalizar un bucle de búsqueda a lo largo de una ocurrencia de conjunto. Esta situación se produce cuando un comando DML que intenta encontrar el siguiente o el anterior miembro de una ocurrencia de conjunto, encuentra que no existe ninguno más. El programa entonces, suele chequear que la variable DB-STATUS tiene el valor EOS para finalizar el loop. De aquí en adelante se asumirá que cuando DB-STATUS = 0 no ha ocurrido ninguna excepción ni ningún error.

9.7 DDL de un sistema en red

El modelo DBTG, como todos los sistemas de bases de datos, posee un sublenguaje de definición datos y de estructuras de datos, el cual permita especificar no sólo los tipos de registro sino además los tipos de conjuntos, su ordenación y sus restricciones.

No obstante, además de no presentar demasiadas complicaciones, este sublenguaje depende en gran manera del lenguaje anfitrión, por lo que en este texto no se va a exponer ninguna de las variantes que aparecen en los libros de teoría.

9.8 Base de datos de ejemplo

A partir de ahora, tanto para los problemas resueltos como para la parte teórica, se utilizará el esquema en red que aparece en la figura 9.7. Este esquema representa la base de datos de una empresa que contiene información de sus EMPLEADOS, los cuales mediante el conjunto SUBORDINADOS tienen asociados un número de empleados subordinados (SUBORDINADO). Esta base de datos, contiene también información sobre los distintos departamentos (DEPARTAMENTO) existentes en la empresa, cada uno de los cuales tendrá asociado un grupo de empleados mediante el conjunto ESTA-EN.

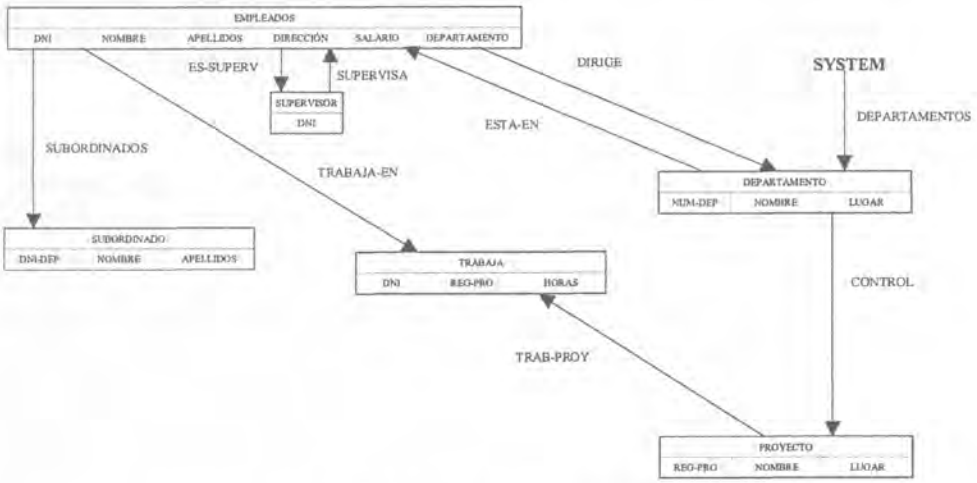


Figura 9.7: Base de datos de ejemplo.

A su vez, mediante el conjunto CONTROL, cada departamento se encontrará trabajando en unos determinados PROYECTOS, los cuales a través de los conjuntos TRAB-PROY y TRABAJA-EN tendrán asociados un grupo de empleados.

Por último, existe un tipo de registro denominado SUPERVISOR, en el cual se encuentran los *dni* de los directores de la empresa. Este tipo de registro está asociado a dos conjuntos distintos; por una parte es propietario del conjunto SUPERVISA lo cual significa que cada SUPERVISOR llevará asociada una lista enlazada de EMPLEADOS a su mando. Además, el tipo de registro SUPERVISOR es miembro del conjunto ES-SUPERV, lo cual significa que cada empleado llevará asociada una lista enlazada de SUPERVISORES, los cuales son jefes suyos.

A la base de datos, se entra mediante el conjunto sistema-propietario DEPARTAMENTOS. Todo esto se verá en los siguientes apartados.

9.9 DML de un sistema en RED

El DML de un sistema en red es el sublenguaje que permite manipular una base de datos de este tipo. Como se mencionó anteriormente, en un sistema DBTG este sublenguaje está embebido en un lenguaje de programación de propósito general llamado lenguaje host. A lo largo de este capítulo se utilizará PASCAL como lenguaje host.

Los comandos o funciones DML pueden ser agrupados en:

- *Funciones de navegación* cuya tarea será moverse y buscar un cierto registro a lo largo de la estructura de conjuntos que componen un sistema en red.
- *Funciones de recuperación* cuya misión será cargar el registro actual
- *Funciones de actualización* las cuales pueden dividirse a su vez en dos subgrupos:
  - Actualización de Registros.
  - Actualización de ocurrencias de Conjuntos.

9.10 Comandos de recuperación y navegación

- *Comandos de recuperación.* El comando para recuperar registros es GET. Este comando lo que hace es cargar el contenido del registro en el cual está posicionado en ese momento el CRU hacia la plantilla de registro.
- *Comandos para localizar registros de un tipo de registro.* Existen dos variaciones sobre el comando o función FIND que permite localizar una ocurrencia de registro de un cierto registro y que además hace que el CRU apunte hacia él.

FIND ANY <nombre de tipo de registro> [USING <lista de campos>]

FIND DUPLICATE <nombre tipo de registro> [USING <lista de campos>]

Se explicará el uso de dichos comandos mediante ejemplos.

♦ Ejemplo 9.1

Usando el esquema de la base de datos mostrada en la figura 9.7, encontrar el primer empleado cuyo apellido sea 'garcia' e imprimir su salario.

Solución:

```
empleado.apellido := 'garcia'
$FIND ANY empleado USING apellido
if DB-STATUS = 0
      then begin
            $GET empleado

            writeln (empleado.salario)
            end
      else writeln ('Registro no encontrado')
```

El funcionamiento de este programa es el siguiente: a la variable *empleado* (que previamente ha debido ser definida en el lenguaje anfitrión) se le asigna en su campo *apellido* el valor 'garcia'. Posteriormente, el comando FIND ANY encuentra el primer registro de la base de datos del tipo EMPLEADO, cuyo campo APELLIDO coincide con el valor del campo *apellido* de la variable *empleado*.

Si la variable DB-STATUS toma el valor cero, entonces la búsqueda ha sido satisfactoria con lo cual se imprimirá el campo SALARIO de la primera ocurrencia de registro del tipo de registro EMPLEADO de la base de datos, cuyo APELLIDO sea 'garcia'.

En caso de que en la base de datos no exista una ocurrencia de registro que cumpla esa condición, entonces la variable DB-STATUS no tomará el valor cero, con lo cual se imprimirá el mensaje de 'Registro no encontrado'.

Pero ¿qué sucede si existe más de una ocurrencia de registro en la base de datos que cumpla esa condición? ¿Cómo pueden imprimirse todos los registros?. La solución se puede encontrar en el siguiente ejemplo:

♦ Ejemplo 9.2

Encontrar todos los empleado cuyo apellido sea 'garcia' e imprimir su salario.

Solución:

```
empleado.apellido := 'garcia';
$FIND ANY empleado USING apellido;
WHILE DB-STATUS = 0 DO
      begin
            $GET empleado;
            writeln (empleado.salario);
            $FIND DUPLICATE empleado USING apellido
            end;
      else writeln ('Registro no encontrado');
```

El funcionamiento de este programa es extremadamente sencillo. A la variable *empleado* (que previamente ha debido ser definida en el lenguaje anfitrión), se le asigna en su campo *apellido* el valor 'garcia'. Posteriormente, el comando FIND ANY encuentra el primer registro de la base de datos del tipo EMPLEADO, cuyo campo APELLIDO coincide con el valor del campo *apellido* de la variable *empleado*.

Si la búsqueda tiene éxito, entonces DB-STATUS toma el valor cero y el programa entra en el bucle WHILE con lo que se carga ese primer registro y se imprime su salario.

Ahora llega el comando FIND DUPLICATE, que lo que hace es encontrar *el siguiente* registro, desde donde se encuentra el CRU que satisface el criterio de búsqueda. Si este comando encuentra otro registro que satisface la condición (DB-STATUS=0), entonces volverá a repetirse todo el proceso hasta que no quede ningún registro cuyo apellido sea 'garcia'.

Es importante recordar que el comando FIND ANY comienza la búsqueda desde el principio de la base de datos y FYND DUPLICATE desde donde está en ese momento el CRU.

- *Comandos para procesar conjuntos*. Para procesar conjuntos se tienen las siguientes variaciones del comando FIND

FIND ANY (FIRST | NEXT | PRIOR | LAST)<nombre de tipo de registro> WITHIN <nombre tipo conjunto> [USING <lista de campos>]

FIND OWNER WHITIN <nombre tipo de conjunto>

En general, una vez que se ha establecido un tipo de conjunto como actual, se puede utilizar el comando FIND para localizar varios registros que participan de una ocurrencia de conjunto, es decir, esta variación del comando FIND va a permitir moverse por la lista enlazada (o doblemente enlazada) que compone una ocurrencia de conjunto, con su registro propietario y una serie de registros miembro, y que además suele estar ordenada según algún campo.

Se estudiará el funcionamiento de estas funciones mediante la solución de los siguientes ejemplos:

♦ Ejemplo 9.3

Encontrar e imprimir el salario de todos los empleados que trabajan en el departamento de nombre 'ventas'.

Solución:

```
departamento.nombre := 'ventas';
$FIND ANY departamento USING nombre;
IF DB-STATUS = 0 THEN
      Begin
      $FIND FIRST empleado WITHIN trabaja-en
      WHILE DB-STATUS = 0 DO
```

```
        begin
            $GET empleado;
            writeln (empleado.salario);
            $FIND NEXT empleado WITHIN trabaja-en
        end;
end;
```

El funcionamiento del programa es el siguiente: mediante la función FIND ANY, se encuentra el primer departamento (se supone que sólo hay uno) cuyo nombre es 'ventas'. Si el resultado es satisfactorio (DB-STATUS = 0) entonces es que se habrá encontrado dicho registro y por tanto el CRU estará apuntando sobre él. Ahora lo que se debería hacer es, una vez localizado este registro, avanzar por la lista enlazada de la cual él es propietario según el conjunto 'trabaja-en', e ir obteniendo todos los miembros de esa ocurrencia de conjunto hasta dar la vuelta entera a la lista.

Eso es lo que hace la siguiente parte del programa. El comando "$FIND FIRST empleado WITHIN trabaja-en", lo que hace es posicionarse en la primera ocurrencia de registro de esa lista enlazada. Mientras vaya encontrando registros (DB-STATUS = 0), irá cargándolos e imprimiendo su atributo nombre.

Como puede adivinarse del código del programa, el encargado de ir avanzando por la lista enlazada de uno en uno es el comando

```
$FIND NEXT empleado WITHIN trabaja-en
```

Utilizando esta función, es posible moverse a lo largo de una ocurrencia de conjunto utilizando los diversos parámetros como PRIOR (el anterior), LAST (el último) o OWNER (el propietario).

9.11 Comandos de actualización

Los lenguajes de los sistemas de bases de datos en red, también contienen comandos y funciones de mantenimiento y actualización del sistema. En el modelo DBTG se pueden clasificar estos comandos en dos grandes grupos:

- *Mantenimiento y actualización de registros* con los comandos STORE, ERASE y MODIFY.
- *Mantenimiento y actualización de conjuntos* con los comandos CONNECT, DISCONNECT y RECONNECT.

El funcionamiento de estos comandos se hará mediante los problemas resueltos de la siguiente sección

9.12 Problemas resueltos

A continuación se exponen una serie de problemas resueltos de dificultad media, que tratan de abarcar un gran número de casos reales que pueden plantearse a la hora de consultar y mantener un sistema de bases de datos en red.

Todos los problemas están referidos a la base de datos indicada en la Fig 9.7.

♦ Problema 9.1

Introducir un nuevo empleado.

Solución:

```
empleado.dni := 77322023;
empleado.nombre := 'JUAN';
empleado.apellidos := 'GARCIA  MARTIN';
empleado.direccion := 'BRAVO MURILLO  25';
empleado.salario := 3.000.000;
empleado.departamento := 'VENTAS';
$STORE empleado;
```

♦ Problema 9.2

Borrar el empleado del problema anterior

Solución:

```
empleado.dni := 77322023;
$FIND ANY empleado USING dni;
IF DB-STATUS = 0 THEN
      $ERASE empleado;
```

El funcionamiento del código anterior es muy sencillo. Primero se introduce el dato a borrar para después encontrarlo mediante FIND ANY. Si el resultado es satisfactorio (DB-STATUS = 0 y el CRU apunta al registro en cuestión) entonces se borra con el comando ERASE.

♦ Problema 9.3

Borrar el empleado del problema anterior y todos sus subordinados según el conjunto SUBORDINADO.

Solución:

Una variación del comando ERASE es el comando ERASE ALL que permite borrar no solamente el registro en cuestión sino además toda la ocurrencia de conjunto entera de la cual él es propietario.

```
empleado.dni := 77322023;
$FIND ANY empleado USING dni;
IF DB-STATUS = 0 THEN
        $ERASE ALL empleado;
```

♦ Problema 9.4

Modificar el sueldo de todos los empleados del departamento de 'VENTAS' y añadirles un 10%.

Solución:

```
departamento.nombre := 'VENTAS';
$FIND ANY departamento USING nombre;
IF DB-STATUS = 0 THEN
    begin
    $FIND FIRST empleado WITHIN esta-en
    WHILE DB-STATUS = 0 DO
        begin
            $GET empleado;
            empleado.salario := empleado.salario * 1.1;
            $MODIFY empleado;
            $FIND NEXT empleado WITHIN esta-en
        end;
    end;
```

Como puede verse, el funcionamiento de este programa es prácticamente idéntico al ejemplo c. La única diferencia es que una vez cargado el valor del registro mediante el comando GET, el valor de su campo salario se multiplica por 1.1 (que es lo mismo que aumentarle el 10%) para posteriormente transferirlo al registro en cuestión mediante el comando MODIFY.

♦ Problema 9.5

Para cada departamento existente en la base de datos, obtener el dni de todos los empleados que trabajan en cada uno de los proyectos asignados.

Solución:

La idea para resolver este problema es la siguiente: primero realizar un bucle que vaya localizando y cargando todos los departamentos de la base de datos. Para cada uno de estos departamentos, realizar otro bucle de manera que se localicen y carguen todos los proyectos de los cuales el departamento es propietario según el conjunto CONTROL. Igualmente, para cada proyecto se realiza otro bucle que recorre y carga todos los miembros de la lista enlazada de la cual el proyecto es propietario.

```
$FIND ANY departamento;
WHILE DB-STATUS = 0 DO
      begin
            $GET departamento;
            writeln (departamento.nombre);
            $FIND FIRST proyecto WITHIN control;
            WHILE DB-STATUS = 0 DO;
             begin
                  $GET proyecto;
                  writeln (proyecto.nombre);
                  $FIND FIRST trabaja WITHIN trab-proy;
                  WHILE DB-STATUS = 0 DO
                        begin
                              $GET trabaja;
                              writeln (trabaja.dni);
                              $FIND NEXT trabaja WITHIN trab-proy;
                        end;
                  $FIND NEXT proyecto WITHIN control;
            end;
            $FIND DUPLICATE departamento;
end;
```

Al no haber puesto ninguna condición en la orden FIND ANY depertamento, se selecciona el primer departamento almacenado en la base de datos.

♦ Problema 9.6

Para cada departamento existente en la base de datos, obtener el dni del empleado que lo dirige.

Solución:

La manera de resolver este problema, es primeramente realizar un bucle de búsqueda y carga de todos los departamentos existentes en la base de datos. Para cada uno de estos departamentos habrá que buscar el propietario de la ocurrencia de conjunto a la que pertenecen según el conjunto DIRIGE. Este propietario es un registro de tipo EMPLEADO que será quien dirige el departamento.

La forma de realizar esta acción en el modelo DBTG es utilizar el comando FIND OWNER WITHIN <nombre tipo conjunto> el cual recorre la lista enlazada hasta encontrar y posicionarse en el registro propietario de la misma.

```
$FIND ANY departamento;
WHILE DB-STATUS = 0 DO
      begin
            $GET departamento;
            writeln (departamento.nombre);
            $FIND OWNER WITHIN dirige;
            IF DB-STATUS = 0 THEN
              $GET (empleado)
              writeln (empleado.dni);
            $FIND DUPLICATE departamento;
end;
```

♦ Problema 9.7

Imprimir el nombre de todos los empleados que trabajan 40 horas dentro del proyecto de nombre 'PROYECTO X'.

Solución:

La forma de abordar este problema es similar a las anteriores (a estas alturas el lector seguramente se ha percatado de lo sencillo que es realizar consultas en el modelo DBTG).

Lo primero que habrá que hacer será localizar el proyecto cuyo nombre es 'PROYECTO X'. Una vez realizado eso, habrá que recorrer la ocurrencia del conjunto TRAB-PROY de la cual él es propietario, para localizar aquellos registros tipo TRABAJA cuyo campo HORAS contenga un valor superior a 40. Para cada uno de estos registros, habrá ahora que encontrar el propietario de la ocurrencia del conjunto TRABAJA-EN en la cual se encuentran. Estos

registros propietarios contendrán la información de los empleados que cumplen con la condición impuesta.

```
Proyecto.nombre := 'PROYECTO X';
$FIND ANY proyecto USING nombre;
IF DB-STATUS = 0 THEN
begin
        trabaja.horas := 40;
        $FIND FIRST trabaja WITHIN trab-proy USING horas;
        WHILE DB-STATUS = 0 DO
    begin
        $GET trabaja;
        $FIND OWNER WITHIN trabaja-en;
        $GET empleado;
        writeln (empleado.nombre);
        $FIND NEXT trabaja WITHIN trab-proy USING horas;
        end
end
```

♦ Problema 9.8

Imprimir el nombre de todos los proyectos en los que 'JUAN GARCIA' trabaja más de 40 horas.

Solución:

```
empleado.nombre := 'JUAN';
empleado.apellido := 'GARCIA';
$FIND ANY empleado USING nombre, apellido;
IF DB-STATUS = 0 THEN
        begin
        $FIND FIRST trabaja WITHIN trabaja-en;
        WHILE DB-STATUS = 0 DO

                begin
                $GET trabaja;
                IF trabaja.horas > 40 THEN
                        begin

                                $FIND OWNER WITHIN trab-proy;
                                $GET proyecto;
                                writeln (proyecto.nombre);
                        end;
```

```
            $FIND NEXT trabaja WITHIN trabaja-en;
            end
    end;
```

El código anterior opera de la siguiente manera: primero asigna a la variable de registro los valores 'JUAN' y 'GARCIA' en sus campos nombre y apellido.

El siguiente paso, es encontrar el registro tipo EMPLEADO cuyos campos nombre y apellido coincidan con los anteriores. Localizado este registro, lo que habrá que hacer ahora es moverse uno a uno por los elementos miembros de la ocurrencia de conjunto TRABAJA-EN de la cual este registro es propietario, y cargar y parar en aquellos cuyo campo horas sea mayor que 40.

Para cada uno de estos registros, habrá que encontrar cuál es su propietario (el proyecto al cual pertenece) e imprimir su nombre. Esto es lo que se realiza con la secuencia de comandos

```
$FIND OWNER WITHIN trab-proy;
$GET proyecto;
writeln (proyecto.nombre);
```

Como puede apreciarse de éste y de los ejemplos anteriores, la manera de realizar búsquedas en un sistema DBGT es siempre la misma; encontrar un cierto registro y a partir de ahí moverse por el enmarañado de conjuntos utilizando las funciones que el sistema provee para tal fin.

♦ Problema 9.9

Para cada departamento, imprimir el nombre de éste y el nombre de su jefe; y para cada empleado que trabaja en ese departamento, imprimir su nombre y la lista de nombres de proyectos en los que el empleado trabaja.

Solución:

Este es un problema NO más complicado que los anteriores sino en todo caso, un poco más largo. La mejor manera de pensarlo es utilizando el esquema de la base de datos y trazar gráficamente el rumbo que el programa debe seguir, ya que las ideas principales siguen siendo las mismas que se expusieron en problemas anteriores.

El código del programa que realiza esta consulta es el siguiente:

```
$FIND FIRST departamento WITHIN departamentos;
WHILE DB-STATUS = 0 DO
        begin
        $GET departamento;
        writeln (departamento.nombre);
        $FIND OWNER WITHIN dirige
        IF DB-STATUS = 0 THEN
                begin
                $GET empleado;
                writeln (empleado.nombre)
                end
        $FIND FIRST empleado WITHIN esta-en;
        WHILE DB-STATUS = 0 DO
                begin
                $GET empleado;
                writeln (empleado.nombre);
                $FIND FIRST trabaja WITHIN trabaja-en;
                WHILE DB-STATUS = 0 DO
                        begin
                        $FIND OWNER WITHIN trab-proy
                        $GET proyecto;
                        writeln (proyecto.nombre);
                        $FIND NEXT trabaja WITHIN trabaja-en;
                        end;
                $FIND NEXT empleado WITHIN esta-en
                end;
        $FIND NEXT departamento WITHIN departamentos
        end;
```

En este problema puede observarse que para localizar todos los departamentos se ha utilizado el conjunto sistema-propietario DEPARTAMENTOS. Podrían haberse buscado también los departamentos como en los problemas anteriores.

El funcionamiento en términos globales del programa es el siguiente; mediante el conjunto sistema-propietario DEPARTAMENTOS se establece un bucle para posicionarse y cargar todos los existentes en la base de datos.

Para cada departamento accedido y cargado, se realiza primero una búsqueda del propietario del conjunto DIRIGR en el cual se encuentra ese departamento, es decir, el registro con los datos del jefe del mismo.

Realizada esta acción, se comienza ahora a buscar todos los empleados que trabajan en este departamento mediante el conjunto ESTA-EN. Para cada uno de estos empleados (propietarios del conjunto TRABAJA-EN), se realiza una navegación por todos los miembros de ese conjunto y por cada uno de ellos

localizado (registros TRABAJA), se localiza su propietario, el cual contiene los datos del proyecto en el que trabaja el empleado.

♦ Problema 9.10

Calcular el número de empleados que son supervisores en cada departamento y su salario medio.

Solución:

Debido a que el DML de un sistema DBTG es un lenguaje que procesa un registro cada vez (no como SQL, que procesa muchos a la vez con una sola orden), es necesario usar toda la potencia del lenguaje host (en este caso PASCAL) para poder realizar ciertas tareas.

En este caso se pide calcular un valor medio, función que sin embargo el DML de un sistema en red no lleva implementada. Es importante llamar la atención en este punto al lector para recordarle otra vez, que lenguajes como SQL o QBE SI llevan implementadas todas estas funciones, y por tanto el usuario o programador no debe preocuparse en ningún momento de cómo construirlas.

Veamos entonces cómo construir esta consulta aprovechado las ventajas que en este caso ofrece PASCAL.

```
$FIND FIRST departamento WITHIN departamentos;
WHILE DB-STATUS = 0 DO
  begin
      $GET departamento;
      write (departamento.nombre);
      SalarioTotal := 0;
      NumdeSupervisores := 0;
      $FIND FIRST empleado WITHIN esta-en;
      WHILE DB-STATUS = 0 DO
            begin
            $GET empleado;
            $FIND FIRST supervisor WITHIN es-superv;
            (* el empleado será un supervisor si es propietario de un registro
            SUPERVISOR via el conjunto ES-SUPERV*)
            IF DB-STATUS = 0 THEN
                  begin
                  SalarioTotal:=SalarioTotal+empleado.salario;
                  NumdeSupervisores:=NumdeSupervisores + 1;
```

```
            end
            $FIND NEXT empleado WITHIN esta-en;
        end;
        writeln('Numero de supervisores=',NumdeSupervisores);
        writeln('Media de salario=',SalarioTotal/NumdeSupervisores);
        writeln();
        $FIND NEXT departamento WITHIN departamentos;
end;
```

Tal y como puede apreciarse en el resultado anterior, el número de supervisores y la media de los salarios han sido calculados mediante dos variables que se han ido modificando durante la ejecución del bucle.

Finalmente, la media ha sido calculada al final mediante la instrucción

```
SalarioTotal / NumdeSupervisores
```

la cual como puede apreciarse, calcula la media mediante la división de la suma de todos los salarios de todos los supervisores, entre el número de supervisores encontrados.

Según se ha especificado en el código, un empleado es supervisor si tiene algún registro miembro dentro de la ocurrencia del conjunto ES-SUPERV del cual él sea propietario.

Capítulo 10
Modelo Jerárquico

10.1 Introducción

A diferencia del modelo de datos relacional, que se fundamenta firmemente en las matemáticas y del modelo en red, que se desarrolló para establecer estándares detallados, el modelo jerárquico se ha desarrollado a partir de la práctica.

Debido a que el modelo jerárquico no tiene un estándar, su estudio requiere un examen de todos los sistemas existentes en la práctica lo cual, queda fuera del propósito de este texto. Afortunadamente, las implementaciones de bases de datos jerárquicas están dominadas por un sistema, *IMS* (Sistema de Gestión de Información de IBM) y debido a ello, este será el sistema que se tomará como estándar en este capítulo. No obstante, existen en la actualidad otros sistemas jerárquicos como *TDMS*, *MARK IV* y el *System 2000.*

Un sistema jerárquico como IMS, presenta a la hora de manejarlo, algunas características un tanto peculiares, sobre todo, en lo referente a su DLL.

En un sistema relacional, como por ejemplo DB2, existe un lenguaje de comandos (SQL) que permite manipular completamente la base de datos, bien de manera interactiva o inmerso desde un lenguaje anfitrión (COBOL, PL/I, etc.). En IMS no existe esta posibilidad, ya que no hay un modo de consulta interactivo. Además, los comandos son realmente funciones que el sistema suministra a modo de librerías y que deben de utilizarse siempre desde un lenguaje anfitrión.

En nuestro caso, al igual que en el capítulo 9, utilizaremos PASCAL como lenguaje anfitrión debido fundamentalmente a que se trata de un lenguaje ampliamente conocido por cualquier estudiante de informática.

10.2 Estructura jerárquica de datos

Una base de datos jerárquica, se compone de un conjunto ordenado de árboles o dicho de manera más precisa, un conjunto ordenado formado por múltiples ocurrencias de un solo tipo de árbol.

Un *Tipo de Arbol,* consiste en un solo tipo de registro raíz, junto con un conjunto ordenado de cero o más tipos de subárboles dependientes (de nivel más bajo).

Como ejemplo, supóngase una base de datos académica, la cual contiene información acerca del sistema de formación interno de una compañía industrial grande.

Figura 10.1: Ejemplo de tipo de registro jerárquico.

Esta compañía, tiene un departamento de formación interna cuya función es impartir cursos a sus empleados. Cada curso se imparte en distintos lugares de la organización y la base de datos contiene detalles tanto de los cursos ya impartidos, como de los ofrecimientos de los cursos que se impartirán en el futuro. Además, contiene en el tipo de registro REQ, los requerimientos necesarios para poder asistir a ese curso.

Este tipo de árbol, contiene un registro raíz denominado CURSO y dos tipos de subárboles cuyas raíces son REQ y OFRECIMIENTO.

Una ocurrencia del árbol principal, podría ser la que aparece en la figura 10.2.

La explicación del árbol mencionado es la siguiente: Por definición, este árbol contiene una sola ocurrencia de CURSO (ya que es el registro raíz). Este CURSO, contiene dos requisitos necesarios para poderse cursar y se imparte en tres fechas distintas y en tres lugares distintos.

En el caso del curso que se ha impartido en 'Valencia', han asistido tres alumnos y ha estado a cargo del profesor de nombre 'Juan'.

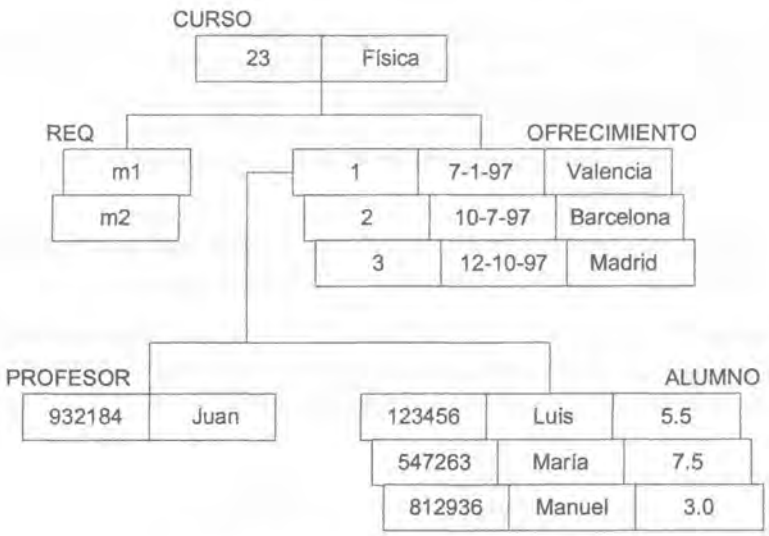


Figura 10.2: Ocurrencia de arbol.

La base de datos completa, estará compuesta por un cierto número de registros raíz, de los cuales estarán colgando todos los subárboles con sus registros asociados. Es importante volver a llamar la atención de que en cada ocurrencia de árbol, sólo puede existir un registro raíz.

Visto ya, de manera muy simplificada, como se organizan los datos dentro de una base de datos jerárquica, pasaremos a describir algunos de los términos que se utilizan de manera habitual dentro de un sistema como IMS.

10.3 Nomenclatura adicional

En una base de datos jerárquica, existen algunos términos que no se utilizan en el resto de los sistemas, pero que forman parte del vocabulario habitual dentro de un entorno como este.

- *Segmento* es la manera en la que se denomina a un registro en la terminología IMS.
- *Segmento raíz* es el segmento mas alto dentro del árbol jerárquico. Sólo puede existir una ocurrencia de segmento raíz por cada árbol.
- *Segmento padre* es cualquier tipo de segmento que tiene colgado de él algún otro tipo de segmento. En el caso del ejemplo anterior, OFRECIMIENTO es segmento padre de ESTUDIANTE y de PROFESOR.

- *Segmento hijo* es cualquier tipo de segmento que tiene un segmento padre asociado. En el ejemplo anterior, PROFESOR y ESTUDIANTE son segmentos hijo de OFRECIMIENTO.
- *Registro de la base de datos* es el término que se utiliza para denominar todo un árbol de segmentos.
- *Segmentos gemelos*. Se denominan así, a todas la ocurrencias de un tipo de segmento que comparten la misma ocurrencia padre.
- *Secuencia jerárquica* es el secuencia ordenada de segmentos de izquierda a derecha y de arriba debajo de un registro de la base de datos. En el caso del ejemplo anterior, la secuencia jerárquica sería:

```
CURSO          23
REQ            M19
REQ            M16
OFRECIMIENTO   1
PROFESOR       932184
ESTUDIANTE     123456
ESTUDIANTE     547263
ESTUDIANTE     812936
OFRECIMIENTO   2
OFRECIMIENTO   3
```

10.4 Area de trabajo de programa

Cualquier programa que se ejecuta en un sistema jerárquico, está compuesto de una serie de sentencias. Algunas de estas son propias del lenguaje anfitrión utilizado (que en nuestro caso será PASCAL) y otras serán llamadas a las funciones propias IMS de manipulación y mantenimiento de la base de datos. Este conjunto de funciones constituyen un sublenguaje denominado DL/I.

Al igual que en las bases de datos en red, el sistema mantiene un *Area de trabajo de programa* para cada programa que se está ejecutando. Este área de trabajo de programa, es un área de almacenamiento de registros intermedios que contiene las siguientes variables.

- *Plantillas de registro* que es el tipo de registro (en el sentido PASCAL) definido por cada tipo de registro existente en la base de datos, y cuya función es poder cargar los datos de la misma.
- *Punteros de actualidad* que son un conjunto de punteros, uno para cada árbol de datos, el cual contiene la dirección del registro en ese árbol en particular al que accedió el programa de aplicación más recientemente.

- *Indicador de estado* que es una variable asignada por el sistema que le indica al programa de aplicación el resultado de la última operación en la base de datos. Esta variable, al igual que en el capítulo anterior, se denomina DB-STATUS y se utiliza el mismo convenio que en el modelo DBTG; si DB-STATUS = 0 la última operación se realizó con éxito.

En el caso del ejemplo expuesto en la figura 10.1, existirá una plantilla de 5 tipos de registros (uno para cada tipo de segmento), 5 punteros de actualidad (un puntero a la última ocurrencia de cada tipo de segmento) y la variable de estado DB-STATUS.

10.5 DDL de un sistema jerárquico

El modelo jerárquico, como todos los sistemas de bases de datos, posee un sublenguaje de definición de datos y estructuras de datos, el cual permita definir no sólo los tipos de registro, sino además los tipos de árboles, su orden, etc.

No obstante, al igual que en el modelo DBTG, este sublenguaje depende en gran manera del lenguaje anfitrión por lo que en este texto no se va a exponer ninguna de las variantes que aparecen en los libros de teoría.

10.6 DML de un sistema jerárquico

El DML de un sistema jerárquico, va a ser la parte del DLL encargada de manipular, actualizar, etc. la base de datos. Al igual que los lenguajes de los sistemas en red, éste sólo puede manejar los registros de la base de datos de uno en uno y no como un todo como hacía SQL o QBE.

Los comandos de este lenguaje, deben estar embebidos dentro de un lenguaje de propósito general denominado lenguaje host.

Utilizaremos a lo largo de este capítulo PASCAL como lenguaje host aunque en la práctica, es más habitual utilizar otro tipo de lenguajes como COBOL.

Antes de seguir, es importante dejar claro que la sintaxis del DML expuesto aquí, puede no coincidir totalmente con la sintaxis de un producto en particular, aunque la manera en la que operan las funciones de todos los sistemas jerárquicos es la misma.

- *El comando GET.* El comando para recuperar registros es GET. Existen diversas formas de aplicar este comando siendo las dos primeras las siguientes:

```
GET FIRST <nombre de tipo de registro> [WHERE <condición>]
```

GET NEXT <nombre tipo de registro> [WHERE<condición>]

La forma más sencilla de recuperar datos, es mediante GET FIRST, el cual siempre comienza a realizar la búsqueda desde el comienzo de toda la base de datos, hasta que encuentra un <nombre de tipo de registro> que satisface <condición>]

Si la búsqueda finaliza con éxito (DB-STATUS = 0), este registro será el registro actual de la base de datos, de la jerarquía y del tipo de registro y además se cargará en la correspondiente variable de programa. Veamos un ejemplo de ello:

- **Ejemplo 10.1**

Encontrar el primer ofrecimiento cuya ciudad sea 'Madrid' e imprimir su fecha.

Solución:

```
$GET FIRST ofrecimiento WHERE ciudad = 'Madrid'
IF DB-STATUS = 0 THEN
writeln (ofrecimiento.fecha)
```

El sistema buscará desde el principio de la base de datos el primer registro tipo OFRECIMIENTO' cuyo campo ciudad tenga el valor 'Madrid'. Si la búsqueda ha sido satisfactoria, entonces imprimirá su campo fecha.

Si existe más de un registro en la base de datos que cumple una determinada condición, sería interesante poderlos recuperar todos. Eso es imposible con el comando GET FIRST ya que siempre comienza la búsqueda desde el principio de la base de datos y por tanto, siempre localizaría el mismo registro.

Para ello existe el comando GET NEXT, el cual comienza a realizar la búsqueda desde el 'registro actual' del tipo de registro especificado en el comando.

Por ejemplo:

- **Ejemplo 10.2**

Encontrar todos ofrecimientos cuya ciudad sea 'Madrid' e imprimir su fecha.

Solución:

```
$GET FIRST ofrecimiento WHERE ciudad = 'MADRID';
WHILE  DB-STATUS = 0 DO
        begin;
        writeln (ofrecimiento.fecha);
```

```
$GET NEXT ofrecimiento WHERE ciudad = 'MADRID'
end;
```

La manera de operar del programa es la siguiente, con el primer comando GET se realiza una búsqueda desde el principio de la base de datos de los ofrecimientos cuya ciudad sea 'Madrid'. Si la búsqueda tiene éxito, entonces este se convierte en el registro actual y se imprime. El comando siguiente GET NEXT, busca ahora el siguiente registro tipo ofrecimiento cuya ciudad sea 'Madrid', pero desde el registro actual de tipo ofrecimiento, es decir, el primero que se ha recuperado.

Si la búsqueda tiene éxito otra vez, entonces este nuevo registro localizado, se convierte en el registro actual, se imprime y se continúa la búsqueda.

También es posible utilizar el comando GET NEXT sin ninguna opción de comparación como por ejemplo:

♦ Ejemplo 10.3

Encontrar todos ofrecimientos e imprimir su fecha.

Solución:

```
$GET FIRST ofrecimiento;
WHILE  DB-STATUS = 0 DO
        begin;
        writeln (ofrecimiento.fecha);
        $GET NEXT ofrecimiento
        end;
```

Al no haber especificado ninguna condición WHERE, todas las ocurrencias de segmento ofrecimiento son válidas.

- *Comandos GET PATH y GET NEXT WITHIN PARENT.* Hasta aquí se han realizado recuperaciones de registros únicos utilizando el comando GET. Cuando se tiene que localizar un registro dentro de una jerarquía, donde varios registros a lo largo de la jerarquía deben de cumplir ciertas condiciones, la recuperación de éste se podría realizarse mediante una serie de condiciones sobre registros a lo largo de todo el camino jerárquico. Para hacer realidad esto, se introduce el siguiente comando:

```
GET (FIRST | NEXT) PATH <camino jerárquico> [WHERE <condición>]
```

donde <camino jerárquico> es una lista de tipos de registros comenzando desde la raíz hasta el destino deseado, y <condición> es una expresión booleana para especificar condiciones de registros individuales a lo largo del camino. Debido a que pueden expresarse varias condiciones, es necesario prefijar los campos sobre

los que se desea imponer una condición con el nombre del tipo de registro al que pertenecen. Veamos un ejemplo:

- **Ejemplo 10.4**

  Imprimir las parejas de curso.titulo y ofrecimiento.fecha de aquellos árboles con un numcurso mayor que 10 y ofrecidos en 'Madrid'

  Solución:

```
$GET FIRST PATH curso, ofrecimiento
WHERE empleado.numcurso >10 AND ofrecimiento.lugar='Madrid';
WHILE DB-STATUS = 0 DO
   begin;
   writeln (curso.titulo, ofrecimiento.fecha)
   $GET NEXT PATH curso, ofrecimiento
    WHERE empleado.numcurso >10 AND ofrecimiento.lugar='Madrid';
   end;
```

  El funcionamiento del programa es el siguiente: se localiza mediante el comando

```
$GET FIRST PATH curso, ofrecimiento
WHERE curso.numcurso >10 AND ofrecimiento.lugar='Madrid';
```

  la primera secuencia jerárquica que cumple estas condiciones. Si la búsqueda ha tenido éxito, el mismo comando GET ha cargado los registros accedidos en la plantilla de registros para posteriormente imprimir los campos deseados mediante la orden

```
writeln (curso.titulo, ofrecimiento.fecha)
```

  Una vez impresa la primera pareja de registros que cumplen esta condición, mediante el comando

```
$GET NEXT PATH curso, ofrecimiento
WHERE empleado.numcurso >10 AND ofrecimiento.lugar='Madrid';
```

  se imprimirá el siguiente camino jerárquico que cumpla las condiciones impuestas. En el momento en el que no exista en la base de datos ningún árbol que cumpla estas condiciones, DB-STATUS = 1 y por tanto el proceso de búsqueda se detendrá.

Otro tipo de problema que suele plantearse en un sistema jerárquico es la de localizar todos los registros de un tipo dado que tienen la misma ocurrencia de segmento padre. En este caso se utiliza el siguiente comando:

```
GET NEXT <tipo de registro hijo>
WITHIN PARENT [<tipo de registro padre>] WHERE <condición>
```

Este comando funciona recuperando la siguiente ocurrencia de tipo de registro hijo, buscando hacia delante desde la ocurrencia de tipo de registro actual y bajo la misma ocurrencia de registro padre.

Como puede observarse, la diferencia fundamental con el comando GET básico es que éste busca en toda la base de datos, mientras que con esta última variación, solo se realiza una búsqueda bajo la ocurrencia de segmento padre actual.

La sentencia <tipo de registro padre> es opcional ya que por defecto el padre es el del <tipo de registro hijo>. Veamos un ejemplo:

♦ Ejemplo 10.5

Imprimir el numofrec de todos los ofrecimientos del curso de título 'Fisica'.

Solución:

```
$GET FIRST PATH curso, ofrecimiento WHERE título = 'Fisica';
(* este comando establece al curso 'Fisica' como padre de la primera
ocurrencia de registro ofrecimiento que cuelga de ella *)
WHILE DB-STATUS=0 DO
    begin
        writeln (ofrecimiento.numofrec);
        $GET NEXT ofrecimiento WITHIN PARENT;
    end;
```

El funcionamiento de la consulta es el siguiente: mediante el comando

```
$GET FIRST PATH curso, ofrecimiento WHERE título = 'Fisica';
```

se localiza el primer árbol con un curso de nombre 'Fisica' y lo hace padre actual de los ofrecimientos. De esta manera, todas las búsquedas de ofrecimiento que se realicen bajo el padre actual, se harán bajo el curso de título 'Fisica' y por tanto, solo se obtendrán las ocurrencias de segmento ofrecimiento que cuelguen directamente de él.

Después de esto, y si la búsqueda ha tenido éxito, se imprimirá el numofrec del primer ofrecimiento que cuelga de este curso. Ahora, con el comando

```
$GET NEXT ofrecimiento WITHIN PARENT;
```

se buscará e imprimirá el siguiente ofrecimiento dentro del mismo curso 'Fisica' hasta que no haya ninguno más. En este momento, el sistema no salta hacia el siguiente árbol de otro curso, sino que la variable DB-STATUS toma un valor distinto de cero y la búsqueda se para.

Antes de seguir, es importante matizar que existen dos métodos para establecer la ocurrencia de segmento que funcionará como padre actual:

- Usando GET FIRST o GET NEXT, el registro recuperado se convierte en el padre actual.
- Usando GET PATH se establece un camino jerárquico de registros padres, de sus respectivos registros hijos.

10.7 Cálculo de funciones de agregados

Debido a que el DML de un sistema jerárquico es un lenguaje que procesa un registro cada vez (no como SQL que procesa muchos a la vez con una sola orden), es necesario usar toda la potencia del lenguaje host (en este caso PASCAL) para poder realizar ciertas tareas como por ejemplo, cálculo de medias, sumas etc.

Veamos un ejemplo:

♦ Ejemplo 10.6

Imprimir el número total de ofrecimientos que tiene el curso de título 'Fisica'.

Solución:

```
$GET FIRST PATH curso, ofrecimiento WHERE título = 'Fisica';
(* este comando establece al curso 'Fisica' como padre de la primera
ocurrencia de registro ofrecimiento que cuelga de ella *)
total := 0;
WHILE DB-STATUS=0 DO
        begin
                total := total +1;
                $GET NEXT ofrecimiento WITHIN PARENT;
        end:
writeln (total)
```

Como puede verse, el funcionamiento del programa es prácticamente igual al del ejemplo anterior salvo una pequeña diferencia; existe una variable (la cual se supone que previamente está definida en el programa principal) denominada *total* que calcula el número de ocurrencias de registro ofrecimiento que existen colgando del curso 'Fisica'.

Es importante ver que en este tipo de sistemas, estas operaciones deben realizarse mediante un programa ya que en el DML no existen funciones del tipo de las que existían en SQL o QBE.

10.8 Actualización de registros.

Los lenguajes de los sistemas jerárquicos de bases de datos, también contienen comandos y funciones de mantenimiento y actualización del sistema. En el sistema IMS se pueden clasificar estos comandos en dos grandes grupos:

- *Mantenimiento y actualización de registros* con los comandos INSERT, DELETE y REPLACE.
- *Retención de registros* con el comando GET HOLD

El funcionamiento de estos comandos se explicará mediante los problemas resueltos del apartado siguiente.

10.9 Problemas resueltos

♦ Problema 10.1

Introducir un nuevo curso llamado 'Quimica' y de numcurso 43.

Solución:

Para insertar un nuevo registro en IMS, se utiliza el comando INSERT. Antes de insertar un registro (o segmento) de un tipo de registro en particular, es necesario primero introducir el valor de los campos en una variable cualquiera (del mismo tipo que el tipo de registro que queremos introducir) del área de trabajo del programa.

Supongamos que tenemos definida una variable del mismo tipo que CURSO denominada i-curso.

```
i-curso.titulo := 'Quimica';
i-curso.numcurso := 43;
$INSERT curso FROM i-curso;
```

Lo que hace este programa, es primeramente cargar en la variable i-curso los valores del registro que se desean introducir. Una vez hecho esto, mediante el comando:

```
$INSERT curso FROM i-curso;
```

se carga el valor de la variable i-curso en el registro curso, creándose por tanto una nueva ocurrencia de este tipo de registro.

Es importante matizar, que si el registro que se desea insertar en la base de datos es un tipo de registro raíz, el comando INSERT creará automáticamente una nueva ocurrencia jerárquica con el nuevo registro como registro raíz.

Si lo que se desea es insertar un registro hijo, es necesario hacer registro actual a su registro padre o bien a alguno de sus hermanos. De esta manera, el comando INSERT insertará el nuevo registro en la ocurrencia de árbol deseada.

♦ Problema 10.2

Introducir un nuevo ofrecimiento del curso de título 'Fisica'.

Solución:

Tal y como se dijo anteriormente, antes de insertar esta nueva ocurrencia, es necesario hacer del curso de título 'FISICA' el padre actual, para lo cual habrá que posicionarse en él o en algún registro del tipo ofrecimiento mediante un comando GET.

Para insertar el nuevo registro, se utilizará la variable i-ofer, la cual se supone que previamente ha sido definida en el programa.

```
$GET FIRST PATH curso, ofrecimiento WHERE título = 'Fisica';
i-ofer.numofrec := 4;
i-curso.fecha := 5-5-98;
i-curso.lugar := 'Madrid'
$INSERT curso FROM i-curso;
```

El funcionamiento del programa es extremadamente sencillo; mediante la orden

```
$GET FIRST PATH curso, ofrecimiento WHERE título = 'Fisica';
```

el sistema se ha posicionado y ha hecho padre actual al curso con título 'Fisica'. También hubiese valido hacer lo siguiente

```
$GET FIRST PATH curso WHERE título = 'Fisica';
```

ya que al haberse posicionado sobre el curso de título 'Fisica' este ya habría sido el padre actual.

Una vez establecido el padre actual, el sistema inserta esta nueva ocurrencia de registro, mediante el comando INSERT, colgando del padre actual es decir, de la ocurrencia de registro con título 'Fisica'.

♦ Problema 10.3

Borrar todos los registros de profesores de nombre 'Pepe'.

Solución:

Para suprimir un registro de la base de datos, es necesario hacerlo primero *registro actual* para después aplicarle el comando DELETE. Para realizar esto, se utiliza el comando GET HOLD el cual hace del registro localizado, el registro actual y además le indica al sistema que el registro será borrado o actualizado.

```
GET  HOLD (FIRST | NEXT <tipo de registro> WHERE <condición>
```

```
$GET HOLD FIRST profesor WHERE nombre = 'Pepe';
WHILE DB-STATUS = 0 DO
        Begin;
        $DELETE profesor;
        $GET HOLD NEXT profesor WHERE nombre = 'Pepe';
end;
```

Es importante a estas alturas, recordar que cuando se borra una ocurrencia de registro de un sistema jerárquico, todas sus ocurrencias de registro hijo también serán suprimidas.

♦ Problema 10.4

Aumentar un punto la calificación de todos los alumnos.

Solución:

Para modificar los valores de un registro, es necesario realizar los siguientes pasos:

- Hacer del registro que va a ser modificado, el registro actual y transferir sus valores hacia la correspondiente variable de programa mediante el comando GET HOLD.
- Modificar los valores en la variable de programa.
- Utilizar el comando REPLACE.

```
$ GET HOLD FIRST alumno;
WHILE DB-STATUS =0 DO
        begin
        i-alumno.calific := i-alumno.calific + 1;
        $REPLACE alumno FROM i-alumno;
        $GET HOLD NEXT alumno;
 end;
```

Al igual que en los problemas anteriores, la explicación del programa es sumamente sencilla: mediante el comando

```
$GET HOLD FIRST alumno;
```

se localiza y carga en la variable de programa el contenido del primer alumno de la base de datos. Una vez realizado esto, se modifica el contenido de la variable de programa con el valor deseado para posteriormente introducirlo mediante el comando

```
$REPLACE alumno FROM i-alumno;
```

Posteriormente, mediante el comando

```
$GET HOLD NEXT alumno;
```

se irán localizando y modificando, todos los alumnos de la base de datos.

♦ Problema 10.5

Imprimir el título de todos los cursos.

Solución:

```
$GET FIRST curso;
WHILE  DB-STATUS = 0 DO
        begin;
        writeln (curso.titulo);
        $GET NEXT curso
        end;
```

El funcionamiento del programa es similar al de los ejemplos antes expuestos. Mediante el comando

```
$GET FIRST curso;
```

se obtiene el primer curso de toda la base de datos. Una vez obtenido e impreso su título, mediante el comando

```
$GET NEXT curso
```

junto con el bucle WHILE se obtienen todos los cursos de la base de datos.

♦ Problema 10.6

Imprimir el título de todos los cursos con numcurso mayor que 24.

Solución:

```
$GET FIRST curso WHERE numcurso > 24;
WHILE  DB-STATUS = 0 DO
        begin;
                writeln (curso.titulo);
                $GET NEXT curso WHERE numcurso > 24;
        end;
```

El funcionamiento del programa anterior, es exactamente igual al del problema 5 salvo la condición de búsqueda de numcurso>24.

♦ Problema 10.7

Imprimir la fecha de todos ofrecimientos de la base de datos cuyo lugar sea 'Madrid'.

Solución:

```
$GET FIRST ofrecimiento WHERE lugar = 'Madrid';
WHILE  DB-STATUS = 0 DO
        begin;
                writeln (ofrecimiento.fecha);
                $GET NEXT ofrecimiento WHERE lugar = 'Madrid';
        end;
```

♦ Problema 10.8

Imprimir el nombre de todos los cursos junto con sus requerimientos y la fecha de sus ofrecimientos asociados.

Solución:

```
$GET FIRST curso
WHILE  DB-STATUS = 0 DO
        begin;
        writeln (curso.titulo);
        (* este comando establece al primer curso como padre de las
        primeras ocurrencias de registro ofrecimiento y req que cuelguen
        de ella *)
                $GET FIRST ofrecimiento WITHIN PARENT;
                WHILE DB-STATUS=0 DO
                begin
                        writeln(ofrecimiento.fecha);
                        $GET NEXT ofrecimiento WITHIN PARENT;
                end:

                $GET FIRST req WITHIN PARENT;
                WHILE DB-STATUS=0 DO
                begin
                        writeln(req.numreq);
                        $GET NEXT req WITHIN PARENT;
                end:
        $GET NEXT curso;
        (* este comando establece al siguiente curso como padre de las
        primeras ocurrencia de registro ofrecimiento y req que cuelguen de
        ella *)
        end;
```

Veamos detenidamente cual es el funcionamiento del programa; el primer comando

```
$GET FIRST curso
```

se posiciona sobre el primer curso de toda la base de datos, y carga en la variable de entorno los valores de sus campos. Si el resultado de la búsqueda ha tenido éxito, entonces imprimirá el nombre de este curso y con el siguiente comando

```
$GET FIRST ofrecimiento WITHIN PARENT
```

localizará el primer registro de tipo ofrecimiento dentro de este árbol (es decir, que cuelga del padre actual que en este momento es el primer registro curso),

cargará los valores de sus campos en la variable de programa asignada e imprimirá su fecha.

Este proceso continuará mediante el comando

```
$GET NEXT ofrecimiento WITHIN PARENT
```

hasta que no quede ningún registro de tipo ofrecimiento colgando de ese registro curso.

Explorada toda la rama de ofrecimiento, el programa realiza ahora la misma acción pero con el tipo de segmento req.

Cuando tampoco quede ninguna ocurrencia de registro de tipo req colgando de este curso, entonces el programa buscará la raíz del siguiente árbol mediante el comando

```
$GET NEXT curso
```

Todo el proceso se repetirá otra vez, hasta que se hayan explorado todos los cursos de la base de datos.

♦ Problema 10.9

Imprimir el nombre de todos los alumnos que han cursado o cursarán los cursos de 'Fisica' o 'Quimica'.

Solución:

La forma de plantear este problema es bastante sencilla. Lo primero que hay que hacer, es buscar todas aquellas ocurrencias de curso, cuyo nombre sea 'FISICA' o 'QUIMICA'. Una vez localizado alguno de los dos, habrá que realizar una búsqueda bajo el padre actual que en este caso se encuentra dos niveles por encima.

```
$GET FIRST curso  WHERE (titulo = 'Fisica' OR titulo = 'Quimica')
WHILE  DB-STATUS = 0 DO
   begin;
     writeln (curso.titulo);
     $GET FIRST alumno WITHIN PARENT curso
     WHILE DB-STATUS=0 DO
     begin
       writeln(alumno.nombre);
       $GET NEXT alumno WITHIN PARENT curso;
```

```
        end;
        $GET NEXT curso WHERE (titulo = 'Fisica' OR titulo = 'Quimica')
    end;
```

Veamos como funciona el código anterior. En primer lugar, mediante el comando

```
$GET FIRST curso WHERE (titulo = 'Fisica' OR titulo = 'Quimica')
```

el sistema se posiciona y carga el primer registro tipo curso cuyo nombre es 'Fisica' o 'Quimica'. Si la búsqueda ha tenido éxito, el programa imprime el nombre del curso y comienza a posicionarse y a cargar, todos los alumnos que cuelgan (aunque dos niveles por debajo) de este curso, mediante el comando

```
$GET FIRST alumno WITHIN PARENT curso
```

Es importante recalcar el hecho de que en este caso, es necesario especificar el padre sobre el cual se desean realizar las búsquedas ya que de otra manera, el sistema buscaría sobre el padre inmediatamente superior, que sería el primer ofrecimiento de ese curso y por tanto, sólo se imprimirían los alumnos que pertenecen a este primer ofrecimiento.

Una vez que se ha obtenido el nombre de uno de los dos cursos a los cuales se hace referencia, el programa mediante

```
$GET NEXT curso WHERE (titulo = 'Fisica' OR titulo = 'Quimica')
```

buscará el otro curso (si es que existe) y realizará las mismas operaciones que sobre el anterior.

Aunque parezca un tanto repetitivo, se vuelve a llamar la atención al lector, sobre el hecho de tener que especificar el padre bajo el cual se desean hacer las búsquedas, cuando este no pertenece a un nivel inmediatamente superior. Si esto no se hiciera, el resultado sería solamente un subconjunto del resultado deseado.

♦ Problema 10.10

Imprimir el nombre de todos los alumnos que han cursado o cursarán los cursos de 'FISICA' y 'QUIMICA'.

Solución:

Este es un problema algo más complicado que los anteriores. La dificultad reside en el hecho de tener que obtener los alumnos que cumplen dos condiciones a la vez.

A diferencia del SQL o el resto de lenguajes relacionales, este tipo de búsquedas no se pueden realizar a la vez sino que de alguna manera, habrá que ir alumno por alumno, y ver si cumple las dos condiciones.

```
$GET FIRST curso WHERE (titulo = 'Fisica')
IF  DB-STATUS = 0 THEN
begin;
 writeln (curso.titulo);
 $GET FIRST alumno WITHIN PARENT curso WHERE (titulo='Fisica');
 WHILE DB-STATUS=0 DO
 begin
   alumn1 := alumno;
   $GET FIRST curso WHERE (titulo = 'Quimica');
   $GET FIRST alumno WITHIN PARENT curso
      WHERE (titulo = 'Quimica');
   WHILE DB-STATUS = 0 DO
   begin
     alumn2 := alumno
     IF (alumn2 = alumn1) THEN writeln (alumno.nombre)
     $GET NEXT alumno WITHIN PARENT curso
     WHERE (titulo = 'Quimica');
   end;
   $GET NEXT alumno WITHIN PARENT curso
      WHERE (titulo = 'Fisica');
 end;
end;
```

Veamos de que manera opera este programa. Primeramente, se supone que existen dos variables de tipo alumno definidas, alumno1 y alumno2. Teniendo en cuenta lo anterior, lo primero que se hace es situarse en el árbol del cual el curso 'FISICA' es raíz con el comando

```
$GET FIRST curso WHERE (titulo = 'Fisica')
```

Localizado este registro, habrá que posicionarse y cargar el primer alumno que está matriculado en este curso con el comando

```
$GET FIRST alumno WITHIN PARENT curso WHERE (titulo = 'Fisica');
```

Localizado este registro, se carga el valor del registro entero en la variable.

El siguiente paso es posicionarse sobre el árbol cuya raíz es el curso 'Quimica' y comenzar a buscar todos los alumnos que de él dependen. Cada uno de estos alumnos se cargan en la variable *alumno2* de manera que si se cumple que *alumno1 = alumno2,* entonces es que este alumno está matriculado en los dos cursos y por tanto, su nombre debe de ser impreso.

El paso siguiente, será localizar el siguiente alumno que se encuentra dentro del árbol cuya raíz es el curso 'Fisica'

```
$GET NEXT alumno WITHIN PARENT curso WHERE (titulo = 'Fisica');
```

y volver a repetir todo el procedimiento.

Es importante recordar, que el sistema conserva un puntero de actualidad, no sólo para el último registro accedido, sino también para cada último registro accedido dentro de cada árbol.

Esto hace posible el que podamos cambiarnos de un árbol a otro, y además seguir avanzando por cada uno de ellos, de manera independiente.

Todos los problemas relacionados con inserciones, modificaciones y actualizaciones, conllevan primero un proceso de búsqueda en el cual radica precisamente la dificultad del problema.

Apéndice A
Base de Datos Automóviles

A.1 Introducción

Es este apéndice se describe la base de datos AUTOMÓVILES a la que se hace referencia en los capítulos 2, 3, 4, 5, 6 y 7. Se trata de una base de datos que contiene información sobre *marcas* de *coches*, los diversos *modelos* que tiene cada *marca*, los *concesionarios* que venden los *coches* y las *ventas* realizadas por estos últimos a los *clientes*.

A.2 Modelo Entidad Relación

Para la base de datos AUTOMÓVILES se propone el siguiente modelo E-R.

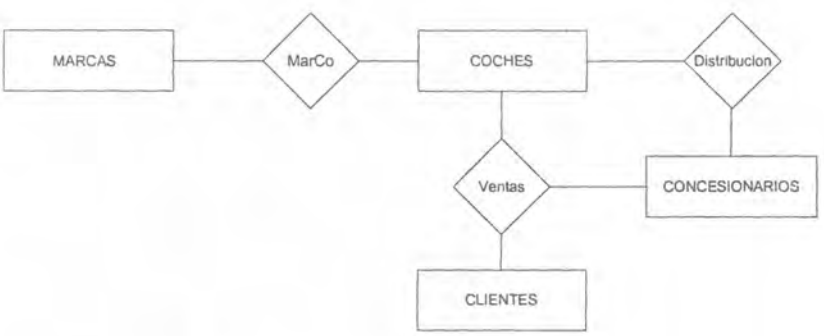


Figura A.1: Modelo Entidad-Relación

A.3 Tablas

El modelo E-R indicado da origen a las siguientes tablas relacionales:

MARCAS (cifm, nombre, ciudad)
COCHES (codcoche, nombre, modelo)
CONCESIONARIOS (cifc, nombre, ciudad)
CLIENTES (dni, nombre, apellidos, ciudad)
DISTRIBUCION (cifc, codcoche, cantidad)
VENTAS (cifc, dni, codcoche, color)
MARCO (cifm, codcoche

las cuales han sido cargadas con los siguientes datos:

MARCAS

cifm	*nombre*	*ciudad*
0001	seat	Madrid
0002	renault	Barcelona
0003	citroen	Valencia
0004	audi	Madrid
0005	opel	Bilbao
0006	bmw	Barcelona

CLIENTES

dni	*nombre*	*apellido*	*ciudad*
0001	Luis	Garcia	Madrid
0002	Antonio	López	Valencia
0003	Juan	Martín	Madrid
0004	María	Garcia	Madrid
0005	Javier	González	Barcelona
0006	Ana	López	Barcelona

VENTAS

cifc	*dni*	*codcoche*	*color*
0001	0001	0001	blanco
0001	0002	0005	rojo
0002	0003	0008	blanco
0002	0001	0006	rojo
0003	0004	0011	rojo
0004	0005	0014	verde

CONCESIONARIOS

cifc	*nombre*	*ciudad*
0001	acar	Madrid
0002	bcar	Madrid
0003	ccar	Barcelona
0004	dcar	Valencia
0005	ecar	Bilbao

COCHES

codcoche	*nombre*	*modelo*
0001	ibiza	glx
0002	ibiza	gti
0003	ibiza	gtd
0004	toledo	gtd
0005	cordoba	gti
0006	megane	1.6
0007	megane	gti
0008	laguna	gtd
0009	laguna	td
0010	zx	16v
0011	zx	td
0012	xantia	gtd
0013	a4	1.8
0014	a4	2.8
0015	astra	caravan
0016	astra	gti
0017	corsa	1.4
0018	300	316i
0019	500	525i
0020	700	750i

MARCO

cifm	*codcoche*
0001	0001
0001	0002
0001	0003
0001	0004
0001	0005
0002	0006
0002	0007
0002	0008
0002	0009
0003	0010
0003	0011
0003	0012
0004	0013
0004	0014
0005	0015
0005	0016
0005	0017
0006	0018
0006	0019
0006	0020

DISTRIBUCION

cifc	*codcoche*	*cantidad*
0001	0001	3
0001	0005	7
0001	0006	7
0002	0006	5
0002	0008	10
0002	0009	10
0003	0010	5
0003	0011	3
0003	0012	5
0004	0013	10
0004	0014	5
0005	0015	10
0005	0016	20
0005	0017	8

Bibliografía

- **Celma M.C.; Casamayor J.C. y Mota L.** "Bases de datos relacionales". Prentice Hall.

- **Connolly T. y Begg C.** "Sistemas de bases de datos". Pearson. Addison Wesley

- **Date C.J.** "Sistemas de bases de datos". 7ª Edición. Prentice Hall.

- **Elmasri y Navathe.** "Sistemas de bases de datos". Addison Wesley

- **Groff J.R. y Weingbert P.** "Manual de referencia SQL". McGraw-Hill.

- **Grau L. y López I.** "Problemas de bases de datos". Sanz y Torres. 2ª Edición.

- **Korth H.F. y Silberchatz** "Fundamentos de Bases de Datos" 4ª Edición. McGraw-Hill.

- **McFadden F. y Hoffer, J.** "Database Management". Third Edition. The Benjamin/Cummings Publishing Company, Inc.

- **Marin F. y Quirós A.** "dBASE IV Técnicas, Aplicaciones y Rutinas de Programación". RA-MA

- **Miguel A.; Piattini M. y Marcos E.** "Diseño de bases de datos relacionales". RA-MA.

- **Miguel A. y Piattini M.** "Fundamentos y modelado de bases de datos". RA-MA. 2ª Edición.

- **Pérez de Madrid A. y Grau L.** "Sistemas de Bases de Datos. Aplicaciones en dBASE". Primera Edición. UNED. 1994.

- **Pons O.; Marín N. y Medina J.M.** "Introducción a las bases de datos". Thomson.

- **Zloof M.** "Query By Example: A Data Base Language". IBM System Journal, 16/4/1977.

ISBN 84-96094-69-3