



## CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

[www.cef.es](http://www.cef.es)

[info@cef.es](mailto:info@cef.es)

## Índice Tema 9

### Introducción.

1. Concepto de base de datos.
2. La arquitectura ANSI/SPARC a tres niveles.
3. El modelo de datos relacional.
  - 3.1. Bases de datos relacionales.
4. Conceptos asociados al modelo relacional.
  - 4.1. Estructuras de datos.
    - 4.1.1. Tablas.
    - 4.1.2. Claves.
  - 4.2. Contenido semántico. Restricciones.
    - 4.2.1. Restricciones inherentes. Reglas de Integridad.
    - 4.2.2. Restricciones explícitas o de usuario.
  - 4.3. Operadores asociados al modelo relacional.
    - 4.3.1. Operadores asociados al Álgebra Relacional.
    - 4.3.2. Ejemplos de acceso a bases de datos mediante el Álgebra Relacional.
    - 4.3.3. Operadores del Cálculo Relacional.
5. Teoría de la normalización.
  - 5.1. Dependencia funcional entre atributos.
  - 5.2. Primera forma normal.
  - 5.3. Segunda forma normal.
  - 5.4. Tercera forma normal.
  - 5.5. Otras formas normales.
6. Representación del modelo lógico de datos. El diagrama de estructura de datos.
  - 6.1. Transformación del modelo conceptual a un modelo relacional.
  - 6.2. Ejemplo práctico.

## 7. Diseño lógico.

### 7.1. Diseño lógico estándar.

- 7.1.1. Transformación de dominios.
- 7.1.2. Transformación de entidades.
- 7.1.3. Transformación de atributos.
- 7.1.4. Transformación de interrelaciones N a M, 1 a N y 1 a 1.
- 7.1.5. Transformación de interrelaciones de dependencia y existencia.
- 7.1.6. Transformación de restricciones de entidades o atributos.
- 7.1.7. Transformación de dependencias de identificación y existencia.
- 7.1.8. Transformación de restricciones en las interrelaciones.
- 7.1.9. Transformación de generalizaciones (relaciones ISA).
- 7.1.10. Transformación de la dimensión temporal.
- 7.1.11. Transformación de atributos derivados.
- 7.1.12. Normalización del esquema obtenido.

### 7.2. Diseño lógico específico.

## 8. Diseño físico.

### 8.1. Metodología de trabajo para la obtención del diseño físico.

- 8.1.1. Análisis de las transacciones.
- 8.1.2. Selección de la organización del almacenamiento en memoria secundaria.
- 8.1.3. Creación de los índices secundarios.
- 8.1.4. Realización de agrupamientos de tablas (clustering).
- 8.1.5. Realización de procesos de desnormalización.
- 8.1.6. Determinación del espacio de almacenamiento necesario.

### 8.2. Diseño de los mecanismos de seguridad de los datos.

- 8.2.1. Creación de las vistas de los usuarios.
- 8.2.2. Fijar las reglas de acceso de los usuarios.

### 8.3. Monitorización y ajuste del sistema.

## 9. La gestión de la concurrencia.

### 9.1. Necesidad de gestión de la concurrencia.

### 9.2. Manejo de transacciones en el SGBD.

### 9.3. Problemas de la concurrencia.

- 9.3.1. El problema de la actualización perdida.
- 9.3.2. El problema de la lectura sucia.
- 9.3.3. El problema de la lectura fantasma o lectura no repetible.

### 9.4. Mecanismos de resolución de conflictos.

- 9.4.1. Métodos pesimistas.
- 9.4.2. Métodos optimistas.

### 9.5. Métodos de control de la concurrencia basados en la semántica.



## CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

### TEMA 9

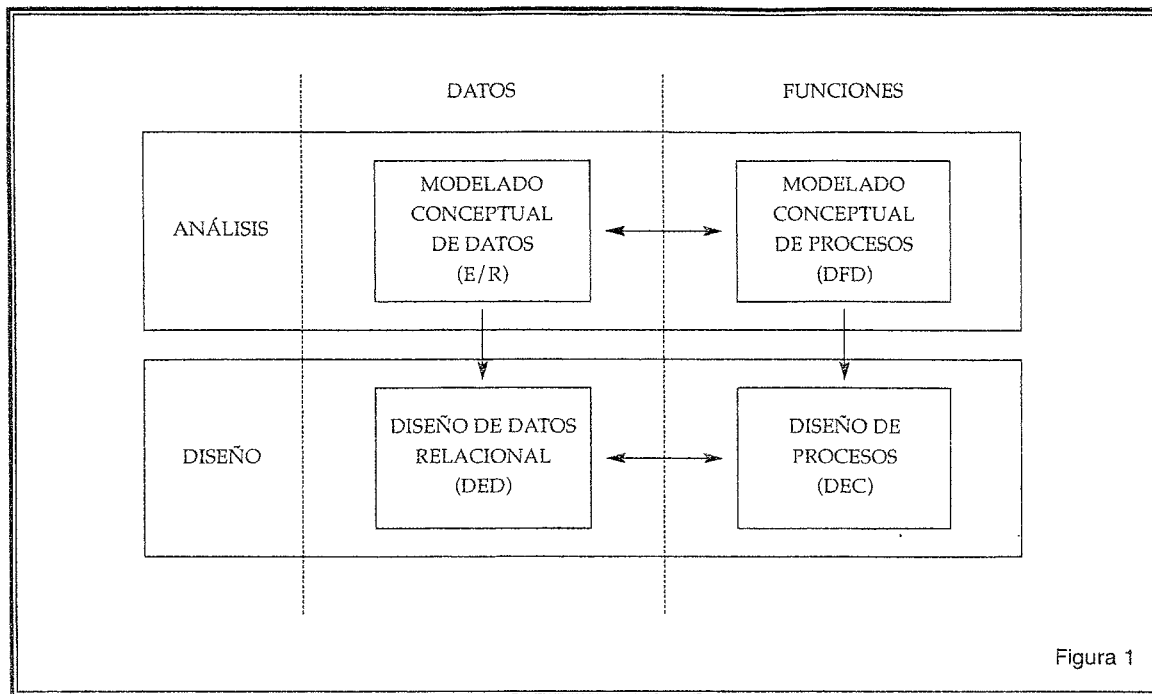
**Diseño de bases de datos. La arquitectura ANSI/SPARC. El modelo lógico relacional. Normalización. Diseño lógico. Diseño físico. Problemas de concurrencia de acceso: lectura sucia, lectura fantasma y bloqueo. Mecanismos de resolución de conflictos.**

#### INTRODUCCIÓN.

Como es sabido, el desarrollo tradicional o estructurado de un Sistema de Información, mantiene una clara separación entre los datos y las funciones o procesos del sistema. La Modelización de Datos es una actividad que se realiza a lo largo del proceso de desarrollo del Sistema de Información en dos momentos diferentes:

1. En el proceso de Análisis, durante el Análisis de Datos, se lleva a cabo el modelado conceptual de datos, cuyo objetivo es captar y representar toda la información del mundo real que es relevante para el sistema, sin considerar cómo se va a implementar éste. Para ello, se parte de la percepción que tiene el usuario del universo del discurso, esto es, el «real percibido» por el usuario, y se representa utilizando una técnica formal (el modelo conceptual de datos, modelo E/R) que tenga una capacidad semántica alta y que sea lo más cercana posible al usuario. Se obtiene así, un esquema conceptual que representa los requisitos del usuario de un modo totalmente independiente de la implementación del sistema.
2. En el proceso de Diseño, durante el Diseño de Datos, debemos ocuparnos de cómo implementar el esquema conceptual obtenido durante el análisis, y para ello se lleva a cabo el modelado lógico de datos durante el diseño lógico de datos, y el modelado físico, durante el diseño físico de datos.

La siguiente figura muestra la situación del diseño de datos en el proceso de desarrollo, y la relación existente entre ésta y otras actividades.



El primer paso del diseño lógico de datos es determinar el modelo de persistencia que se va a utilizar, es decir, el modo en que se almacenarán los datos (sistemas de ficheros o sistemas de bases de datos). Hoy día es difícil pensar en un sistema de información que no utilice bases de datos para almacenar su información, por lo que sólo nos referiremos a este aspecto.

Dependiendo del modelo que soporte el Sistema de Gestión de Base de Datos (SGBD) (modelo Jerárquico, modelo en Red o Codasyl, o modelo Relacional) existen diferentes tipos de bases de datos. Cada modelo tiene sus características propias que lo hace más adecuado para unos problemas u otros y, además, el modelo elegido va a condicionar en gran medida los lenguajes de datos utilizados, ya que la propia estructura del modelo llega a imponer determinadas formas de acceder a los datos.

Nosotros nos vamos a centrar exclusivamente en el Modelo Relacional, que es el más utilizado como técnica de diseño estructurado de datos, y a estos efectos, una vez apuntadas sus características principales, completaremos su concepción refiriéndonos a los tres aspectos básicos que le son propios:

- Las estructuras de datos, esto es, a las tablas o relaciones y a las claves.
- El contenido semántico, es decir, todos aquellos aspectos del universo del discurso que no se pueden modelizar mediante la definición de conjuntos y relaciones, sino que necesitan un nivel superior de descripción que los imbuya de significado, el cual se traduce en el establecimiento de una serie de restricciones (inherentes y explícitas), cuya finalidad es mantener la integridad y validez de los datos almacenados y aumentar el grado de información que proporciona el esquema lógico de datos.
- Los operadores asociados al modelo, tanto del Álgebra Relacional como del Cálculo Relacional, con especial atención a los primeros.

El siguiente objetivo de este tema será el estudio de la teoría de la normalización, como técnica de diseño de bases de datos relacionales consistente en reemplazar un conjunto dado de relaciones por otro conjunto de relaciones que tengan una estructura más simple y más regular.

Posteriormente, referenciaremos la representación gráfica del modelo relacional de datos según la técnica del Diagrama de Estructura de Datos (DED).

A continuación nos adentraremos en el diseño lógico, considerado éste como la etapa del proceso de diseño de una Base de Datos en la que se obtiene la representación de la estructura de la base de datos en términos de almacenamiento (tablas). La obtención de esta estructura implica la aplicación de unas reglas de transformación de los elementos previamente existentes en el modelo conceptual de la BD.

Cómo parte final del diseño de base datos veremos el diseño físico que es la etapa que incluye las acciones de configuración y ajuste del almacenamiento físico y de la seguridad de la BD. El diseño físico es una tarea compleja y dependiente del SGBD utilizados y del uso concreto que se pretenda hacer de la BD diseñada, por lo que en este tema se van a describir la problemática general que se aborda en esta etapa y los criterios de toma de decisiones en esas etapas para resolver un problema.

Se describe en este tema también, el acceso concurrente a las BD y los conflictos que este acceso plantea, así como los mecanismos que los Sistemas Gestores de Bases de Datos (SGBD) utilizan para resolverlos.

## 1. CONCEPTO DE BASE DE DATOS.

Una primera aproximación al concepto de base de datos es definirla como un conjunto, colección o depósito de datos interrelacionados y estructurados de acuerdo con un modelo capaz de recoger el máximo contenido semántico, y almacenarlos en un soporte informático de acceso directo.

Dada la relevancia que tienen en el mundo real las interrelaciones entre los datos, es imprescindible que la base de datos sea capaz de almacenar estas interrelaciones. Por otra parte, existen también restricciones semánticas que, asimismo, deben almacenarse junto con los datos.

La redundancia de los datos debe ser controlada, de forma que no existan duplicidades innecesarias, y que las redundancias físicas, muchas veces convenientes a fin de ganar en eficiencia, sean tratadas por el mismo sistema de modo que no puedan producirse inconsistencias. Es decir, en las bases de datos no debe existir redundancia lógica, aunque sí se admite cierta redundancia física.

Otra característica esencial de las bases de datos es la independencia, tanto física como lógica, entre los datos y los tratamientos, así como el hecho de que las bases de datos han de atender a múltiples usuarios y diferentes aplicaciones; cosas, ambas, que las diferencian esencialmente de los sistemas de ficheros convencionales.

Por último, la definición o descripción del conjunto de datos contenidos en la base (a lo que se denomina estructura o esquema de la base de datos) deben ser únicas y estar integradas con los mismos datos. La actualización y recuperación de los datos deben realizarse mediante procesos bien determinados, incluidos en el SGBD, el cual ha de proporcionar también instrumentos que faciliten el mantenimiento de la seguridad (confidencialidad, disponibilidad e integridad) del conjunto de datos.

En definitiva, de acuerdo con las características que se acaban de citar, y siguiendo a De Miguel y Piattini, se puede definir base de datos como:

«Una colección o depósito de datos integrados, almacenados en soporte secundario (no volátil) y con redundancia controlada. Los datos, que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de ellos, y su definición (estructura de la base de datos) única y

almacenada junto con los datos, se ha de apoyar en un modelo de datos, el cual ha de permitir captar las interrelaciones y restricciones semánticas existentes en el mundo real. Los procedimientos de actualización y recuperación, comunes y bien determinados, facilitarán la seguridad del conjunto de datos».

Un Sistema de Gestión de Bases de datos (SGBD) es un conjunto coordinado de programas, procedimientos y lenguajes, que suministra, tanto a usuarios informáticos como no informáticos, los medios necesarios para describir, manipular y utilizar los datos almacenados en la base de datos, manteniendo su integridad, confidencialidad y seguridad.

El SGBD, esto es, el conjunto de programas que permiten la implantación, acceso y mantenimiento de la base de datos, proporciona interfaces de comunicación entre los usuarios y la base de datos, así como procedimientos para que el administrador de la base realice sus tareas, e incluye:

- Un lenguaje de descripción de datos (LDD), para definir las entidades, su identificación, atributos, interrelaciones, autorizaciones de acceso y restricciones de integridad, así como el espacio físico reservado para la base de datos, la longitud de los campos o elementos de datos, su modo de representación, los caminos de acceso y la correspondencia entre los aspectos lógico y físico.
- Un lenguaje de manipulación de datos (LMD), para llevar a cabo la recuperación y actualización de los datos.

El SGBD, junto con la base de datos y los usuarios, constituye el Sistema de Base de Datos (SBD).

## 2. LA ARQUITECTURA ANSI/SPARC A TRES NIVELES.

Como ya se puso de manifiesto, hoy día las bases de datos son una parte integrante de los sistemas de información y por tanto, al igual que el sistema en sí, deben tener la capacidad de adaptarse a los cambios que se produzcan en el entorno, ya sean de tipo físico (cambios en el hardware, en el formato de los ficheros, etc.) o de tipo lógico (cambios en los tratamientos, en el lenguaje de programación, etc.). Esto se consigue garantizando la independencia de los datos, es decir, la independencia entre su estructura lógica y la forma en que los datos se guardan físicamente. Consecuentemente, uno de los principales objetivos de las bases de datos es conseguir la independencia entre las estructuras lógica y física de los datos puesto que conseguido este objetivo estará garantizada la independencia entre los datos y las aplicaciones, requisito imprescindible para satisfacer las necesidades de cambios, ya sea en los datos o en los requerimientos de los usuarios.

Conscientes de esta necesidad, el Organismo de Estandarización de Estados Unidos ANSI (American National Standards Institute) propuso en 1975 una arquitectura de bases de datos orientada a independizar los datos de los tratamientos, estructurada en tres niveles de abstracción en el Sistema de Gestión de Bases de Datos (SGBD), que proporciona el máximo grado de independencia física/lógica de los datos a nivel de descripción.

ANSI define el concepto de independencia de los datos como la «capacidad de un SGBD para permitir que las referencias a los datos almacenados, especialmente en los programas y en sus descripciones de los datos, estén aisladas de los cambios y de los diferentes usos en el entorno de los datos, como pueden ser la forma de almacenar dichos datos, el modo de compartirlos con otros programas y la manera en que se reorganizan para mejorar el rendimiento del sistema de base de datos».

Esta independencia entre la parte física y la parte lógica de la base de datos se puede entender compuesta por dos formas de independencia complementarias: la independencia en la descripción y la independencia en los tratamientos.

La independencia en la descripción se consigue separando la definición física y la definición lógica de los datos y permitirá que un único diseño lógico de la estructura de los datos se pueda migrar de unos SGBD a otros sin sufrir modificaciones lógicas, es decir, de forma transparente para el usuario final.

La independencia en los tratamientos consiste en separar la forma de almacenar los datos, de las operaciones necesarias para acceder a los mismos. Esta independencia permitirá utilizar los mismos métodos para acceder a los datos con independencia de cómo se almacenen éstos (según un modelo jerárquico, según un modelo relacional, etc.).

Para el logro de la independencia de los datos, ANSI propone que las bases de datos se construyan siguiendo una estructura o arquitectura de tres niveles organizados jerárquicamente. Los tres niveles que propone la arquitectura ANSI son:

1. Nivel Conceptual. Este nivel está orientado hacia la visión lógica del conjunto de información que proviene del mundo real. Es decir, en el nivel conceptual se pretende reflejar la estructura y las relaciones existentes entre los datos del mundo real que se van a almacenar en la base de datos (universo del discurso), aislando entre sí el nivel externo (vista del usuario) y el nivel físico (vista de la máquina).
2. Nivel Lógico global o Externo. Puesto que no todos los programas ni los usuarios que acceden a la base de datos necesitan tener una visión total de la información almacenada en la misma, resulta conveniente crear vistas o esquemas específicos según las necesidades de cada uno. Esta es la razón del nivel externo, guardar las distintas vistas parciales de la base de datos que se muestran a los usuarios. Por tanto, se puede decir que el nivel externo está orientado hacia el usuario y comprende las características lógicas de los datos para los programas de aplicación.

La existencia de este nivel aísla a los usuarios no sólo del aspecto físico de la base de datos, sino también de cualquier parte de la misma que no esté relacionada con la tarea que deben desempeñar, aumentando así la protección frente a cambios en otras áreas de la organización y aumentando la seguridad de los datos, ya que al crearse vistas parciales de la base de datos los usuarios sólo tienen acceso a las partes seleccionadas de la misma.

3. Nivel Físico o Interno. En este nivel se especifica qué, cómo y dónde se van a almacenar los datos físicamente. Está orientado hacia la máquina y comprende las características de tipo físico.

Este nivel se ocupa de tratar con el sistema operativo, con el sistema de ficheros, con los dispositivos de entrada/salida y, en general, con todos aquellos aspectos necesarios para almacenar efectivamente la información en el ordenador. Por tanto, el contenido del nivel físico o interno depende totalmente de la combinación hardware/software que se emplee en cada instalación.

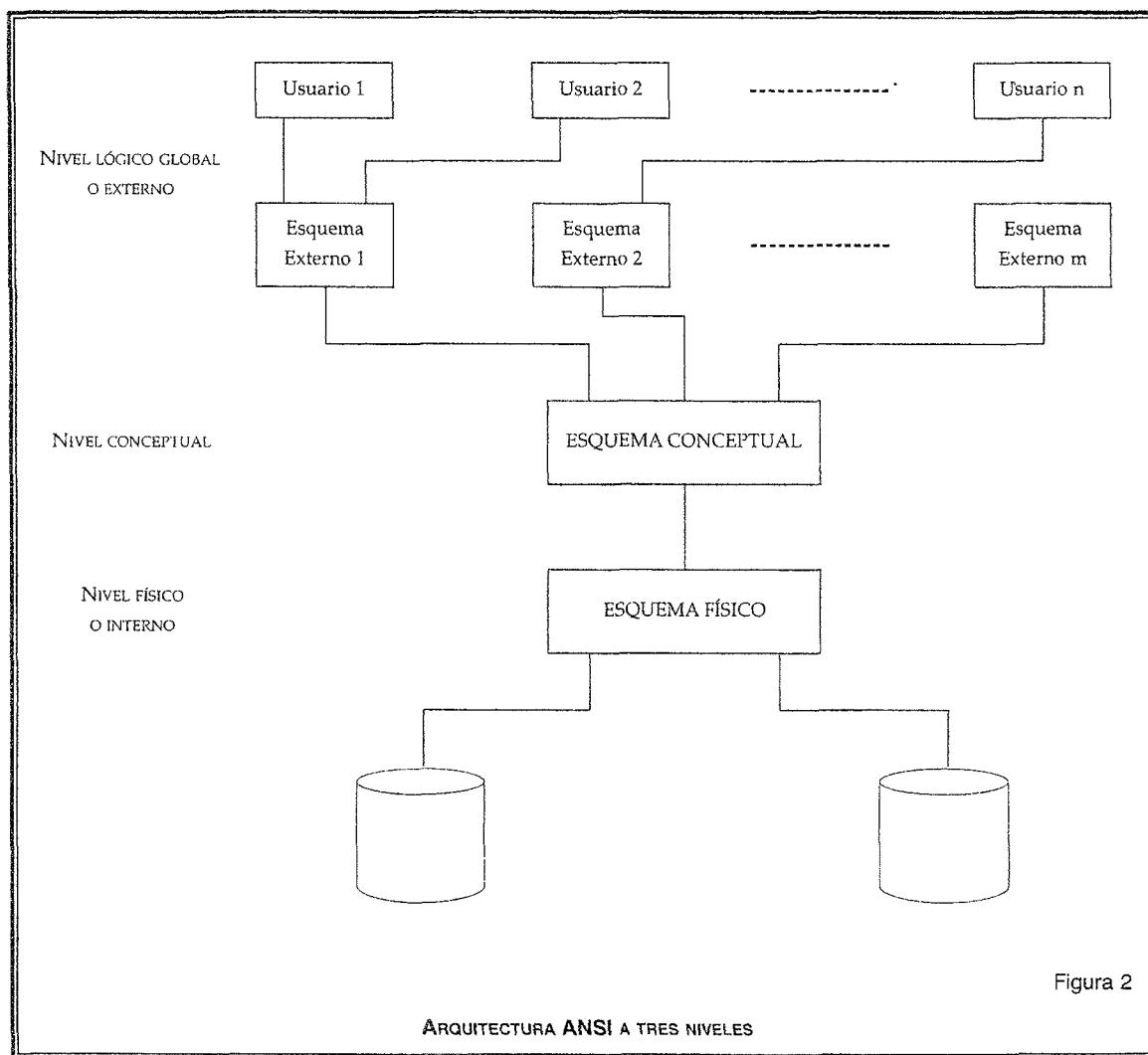
Si definimos Modelo de Datos como el conjunto de conceptos, reglas y convenciones que permiten describir y manipular los datos de la parcela del mundo real que constituye nuestro universo del discurso, al aplicar el Modelo de Datos, a un cierto universo del discurso se obtiene una estructura de datos llamada Esquema.

Existen distintos modelos de datos para construir los esquemas correspondientes a cada nivel de abstracción de la arquitectura ANSI:

- a) El Esquema Conceptual, que debe captar y almacenar el «universo del discurso» que ha de tratar el sistema de información, y sirve de punto de control para futuros desarrollos de la base de datos, aísla la representación de la información de los requerimientos de la máquina y de las

exigencias de cada usuario, e independiza la definición de la información de los SGBD. Es el correspondiente al modelo de datos conceptual.

- b) El Esquema Lógico o Externo, que permite «ver» a cada tipo de usuario de la base de datos, sólo aquella parte que es de su interés. Al haber diferentes usuarios con distintas necesidades, existirán distintos esquemas lógicos o externos para un mismo esquema conceptual. Es el correspondiente al modelo lógico de datos convencional.
- c) El Esquema Físico o Interno. Es el que especifica cómo son almacenados los datos en memoria, describe la estructura de la base de datos en forma de modelo conceptual de almacenamiento y depende del entorno físico donde se vaya a implantar el sistema.



### 3. EL MODELO DE DATOS RELACIONAL.

El Modelo de Datos Relacional es un modelo lógico de datos que fue presentado por Codd en 1970. Es un modelo muy simple, con sólidos fundamentos matemáticos basados en la teoría de conjuntos, cuyo objeto es la representación del universo del discurso mediante los convencionalismos del álgebra relacional.



Es un modelo fácil de manejar por usuarios no especializados y sus características fundamentales son las siguientes:

- Está basado en un modelo matemático con reglas y algoritmos algebraicos establecidos, lo que permite el desarrollo de lenguajes de acceso y manipulación potentes basados, bien en la teoría de conjuntos (álgebra relacional), bien en la lógica de predicados (cálculo relacional).
- Las estructuras de datos son simples. Son relaciones que se presentan en forma de tablas bidimensionales, permitiendo un alto grado de independencia de la información con respecto al medio físico de almacenamiento de los datos. Las tablas representan tanto las entidades del universo del discurso como las relaciones entre las mismas.
- Permite incorporar aspectos semánticos del universo del discurso mediante el establecimiento de reglas de integridad, las cuales garantizan también la consistencia de los datos. Igualmente, el proceso de normalización, al eliminar ciertas anomalías que pueden presentarse en las relaciones, representa una valiosa ayuda para el diseño de la base de datos.

### 3.1. BASES DE DATOS RELACIONALES.

Las bases de datos relacionales se han ido imponiendo a partir de los años 80 debido a la superioridad que ofrece el modelo de datos en que se basan, el modelo relacional, frente a sus antecesores: el modelo jerárquico o de estructuras en árbol y el modelo Codasyl o de estructuras en red.

En el año 1985, Codd estableció una regla general global, llamada «Regla Cero», y doce principios, de los cuales al menos seis deben satisfacerse para que una base de datos pueda considerarse relacional. Estos son los siguientes:

- Regla 0. Gestión de una Base de Datos Relacional. «Un sistema de gestión de bases de datos relacionales (SGBDR) debe ser capaz de manejar las bases de datos exclusivamente con sus capacidades relacionales».
- Regla 1. Representación de la información. «Toda la información en una base de datos relacional se representa explícitamente a nivel lógico y de una manera única, por medio de valores en tablas».
- Regla 2. Acceso garantizado. «Todos y cada uno de los datos elementales en una base de datos relacional deben ser accesibles lógicamente mediante una combinación de: nombre de tabla, valor de clave primaria y nombre de columna».
- Regla 3. Representación sistemática de la información que falta. «Los valores nulos deben ser soportados por un SGBD completamente relacional para representar, de modo sistemático, la información desconocida o inaplicable».
- Regla 4. Catálogo dinámico. «La descripción de la base de datos se representa a nivel lógico en la misma forma que los datos, de forma que los usuarios autorizados puedan aplicar el mismo lenguaje relacional para consultarlo».
- Regla 5. Sublenguaje global de datos. «Debe existir, al menos, un lenguaje cuyas sentencias sean expresables mediante una sintaxis bien definida, como cadena de caracteres, y capaz de soportar definición de datos, definición de vistas, manipulación de datos, restricciones de integridad, autorizaciones y manejo de transacciones».

- Regla 6. Actualización de vistas. «Todas las vistas teóricamente actualizables deberán ser también actualizables por el sistema».
- Regla 7. Inserciones, actualizaciones y eliminaciones de alto nivel. «La capacidad para manejar como un solo operando una relación base o una relación derivada se aplica, no sólo a las recuperaciones de datos, sino también a sus inserciones, actualizaciones y eliminaciones».
- Regla 8. Independencia física de los datos. «Los programas de aplicaciones y las actividades terminales permanecerán lógicamente inalterados siempre que se realicen cambios en las representaciones de almacenamiento o en los métodos de acceso».
- Regla 9. Independencia lógica de los datos. «Cuando se efectúe en las tablas cualquier tipo de cambio que preserve la información, los programas de aplicación permanecerán intactos».
- Regla 10. Independencia de la integridad. «Las reglas de integridad de un base de datos particular deben ser definibles por medio del sublenguaje de datos relacional y almacenables en el catálogo, no en los programas de aplicaciones».
- Regla 11. Independencia de la distribución. «Un sistema relacional debe tener un sublenguaje de datos que pueda soportar bases de datos distribuidas sin alterar los programas de aplicaciones o actividades finales».
- Regla 12. Regla de la no inversión. «Si un sistema relacional tiene un lenguaje de bajo nivel, éste no puede ser utilizado para pasar por alto las reglas de integridad y las restricciones expresadas por medio del lenguaje relacional de más alto nivel».

En definitiva, de las Reglas de Codd podemos concluir que un sistema de base de datos relacional se caracteriza:

- Por presentar externamente sus datos como tablas, aunque internamente se sigan manejando de forma convencional mediante índices, páginas, etc.
- Por la disponibilidad de un lenguaje para operar con las tablas, que al menos tenga las funciones de recuperación, modificación, selección de subconjuntos de tablas sin predefinición de caminos de acceso, definición de datos, etc., y que proporcione medios para controlar la integridad, seguridad y consistencia de los datos.
- Por disponer de interfaces que permitan el acceso concurrente desde terminales interactivos y programas de aplicación, así como herramientas estándar para controlar la operación y facilitar los procesos de respaldo y recuperación.

#### 4. CONCEPTOS ASOCIADOS AL MODELO RELACIONAL.

Como se ha indicado, el Modelo de Datos Relacional o Modelo de Codd se caracteriza por las estructuras de datos que soporta, los operadores asociados y el contenido semántico que se puede incluir en el modelo. En los siguientes apartados estudiaremos los conceptos asociados a cada una de estas facetas, las capacidades de modelización que ofrecen y sus restricciones.

#### 4.1. ESTRUCTURAS DE DATOS.

En el Modelo Relacional los datos se estructuran lógicamente en forma de tablas o relaciones, manteniendo la independencia de esta estructura lógica respecto al modo de almacenamiento y otras características de tipo físico.

##### 4.1.1. Tablas.

La estructura básica del modelo relacional es la tabla, que representa una relación, y en la cual se distinguen los siguientes elementos:

- **Atributos.** Son las columnas de cada tabla y representan las propiedades de las entidades presentes en el universo del discurso que nos interesa almacenar en la base de datos. Cada uno de estos atributos puede tomar valores dentro de un rango determinado denominado «dominio». Varios atributos pueden compartir un único dominio.
- **Tuplas.** Son las filas de cada tabla y representan cada una de las ocurrencias de la relación que representa la tabla.
- **Grado de la relación.** Es el número de atributos o número de columnas de la tabla.
- **Cardinalidad de la relación.** Es el número de tuplas o filas u ocurrencias de la relación.
- **Dominio.** Es el rango de valores aceptable para un atributo dado. Depende exclusivamente del atributo y va a condicionar los valores posibles dentro de cada celda de la tabla. Los valores que forman el dominio deben ser homogéneos, es decir, del mismo tipo, y atómicos, esto es, indivisibles.
- **Descriptor de una tabla o relación.** Es un conjunto no vacío de atributos de la tabla o relación.

Ahora ya, de manera formal podemos definir el concepto de Relación de la siguiente manera: dados los dominios  $D_1, D_2, \dots, D_n$  (no necesariamente distintos), diremos que  $R$  es una relación entre estos «n» conjuntos, si  $R$  es un conjunto de «n» tuplas  $(d_1, d_2, \dots, d_n)$  tal que  $d_1$  pertenece al dominio  $D_1$ ,  $d_2$  pertenece al dominio  $D_2$ , etc.

Reafirmaremos estas definiciones mediante el siguiente ejemplo:

Ejemplo. Sea la relación «Empleado» representada por la siguiente tabla:

<b>ATRIB.</b> <b>TUPLA</b>	<b>DNI</b>	<b>NOMBRE</b>	<b>PROVINCIA</b>	<b>DEPARTAMENTO</b>
1	12.567.421	Lucas	Madrid	Admón.
2	1.457.820	Felipe	Madrid	Produc.
3	2.546.435	Lucas	Sevilla	Ventas
--	--	--	--	--
100	50.215.111	Juan	Orense	Ventas

Se trata de una tabla o relación que tiene 100 filas o tuplas y 4 columnas o atributos. Por tanto, esta relación es de grado 4 y cardinalidad 100.

Cada atributo tiene su dominio. Así, el dominio del atributo DNI será cualquier cadena numérica de 10 cifras y el de los atributos Nombre, Provincia y Departamento será cualquier conjunto de caracteres alfabéticos de determinada longitud.

Es importante destacar que la corrección de los valores de los atributos no se puede garantizar sin la inclusión del contenido semántico en formas de reglas de integridad. Es decir, si no se incluye el contenido semántico, esto es, alguna restricción, nada impediría introducir el valor «000000» como DNI, o «HSJJRKL» como Provincia, ya que a pesar de ser incorrectos, los valores están dentro de los dominios correspondientes.

Las tablas o relaciones del Modelo Relacional poseen las siguientes características:

- Cada tabla debe contener un solo tipo de filas. El formato de cada fila queda definido por el esquema de la relación, esto es, por los nombres de los atributos y su formato. Dicho de otra forma, todas las filas de una tabla están compuestas por el mismo número y tipo de atributos.
- No puede haber filas o tuplas duplicadas.
- El orden de las tuplas es irrelevante.
- El orden de los atributos no es significativo, aunque debe respetarse una vez establecido.
- La tabla es plana; es decir, en el cruce de un atributo con una tupla, o lo que es lo mismo, en el de una columna con una fila, sólo puede haber un valor, que además, debe estar dentro del dominio de la columna correspondiente.

#### 4.1.2. Claves.

Una característica de una tabla es que no puede haber dos filas iguales. Esto obliga, necesariamente, a que haya uno o varios atributos que se puedan utilizar para distinguir unas tuplas de otras, dicho de otra forma, que actúen como claves. Tenemos así, los siguientes conceptos:

- Clave candidata. Es el conjunto no vacío de atributos que identifica unívoca y mínimamente a cada tupla de una tabla. Al decir mínimamente se entiende que si de este conjunto de atributos se elimina uno de ellos, el conjunto resultante deja de ser clave candidata.

Por ejemplo, en la tabla «Empleado» podemos elegir como clave candidata los atributos DNI y Nombre. Si eliminamos el atributo DNI, Nombre deja de ser clave candidata porque no identifica unívocamente a cada tupla de la tabla ya que puede haber nombres repetidos.

- Clave primaria. Es la clave candidata que elige el usuario. La clave primaria puede ser simple, si sólo consta de un atributo, o compuesta, si está formada por más de uno. Los atributos que forman parte de la clave primaria se denominan atributos principales, mientras que los restantes atributos son los no principales.

Por razones de eficiencia, normalmente se suele elegir como clave primaria la más corta y si es posible, simple. En nuestro ejemplo se elegiría el DNI.

Una propiedad fundamental de la clave primaria es lo que se conoce como Integridad de entidad: «Ningún atributo que forme parte de la clave primaria puede tomar un valor nulo, entendiendo por tal, un valor desconocido o inaplicable». En efecto, si algún atributo pudiera adoptar el valor nulo, perdería su capacidad para identificar biunívocamente las tuplas de la relación.

- Claves alternativas. Son las claves candidatas que no han sido elegidas como clave primaria.

#### 4.2. CONTENIDO SEMÁNTICO. RESTRICCIONES.

Se entiende por contenido semántico todos aquellos aspectos del universo del discurso que no se pueden modelizar mediante la definición de conjuntos y relaciones, sino que necesitan un nivel superior de descripción que imbuya de significado a dichos conjuntos y relaciones. De ahí el adjetivo «semántico».

Este nivel superior de descripción del modelo se traduce en el establecimiento de una serie de restricciones adicionales a las propias del modelo relacional, cuya finalidad es mantener la integridad y validez de los datos almacenados y aumentar el grado de información que proporciona el esquema lógico de datos. Dicho de otra forma, las razones que justifican que el modelo requiera restricciones son de dos clases:

- a) Una razón semántica, es decir, permitir al esquema de datos reflejar más exactamente la información a modelar.
- b) Una razón de integridad, esto es, comunicar al SGBD qué estados de la base de datos son permitidos.

Se definen dos tipos básicos de restricciones:

- Restricciones inherentes o estructurales. Son las que forman parte integral de estructura del modelo.
- Restricciones explícitas. Son las definidas por el usuario y el diseñador del esquema de datos.

##### 4.2.1. Restricciones inherentes. Reglas de Integridad.

Las restricciones inherentes son propias del modelo en sí, es decir, forman parte de su definición y su estructura. Por este hecho, algunas de estas restricciones se derivan directamente de las propiedades de la relación matemática, como por ejemplo, la no repetición de tuplas en una tabla o relación, lo que implica, como se ha visto anteriormente, la existencia de un atributo o conjunto de atributos (clave) que identifican a cada tupla de forma única y no redundante.

Por el contrario, otras restricciones inherentes al modelo relacional no se derivan tan directamente como las anteriores de las relaciones matemáticas. Estas restricciones se conocen como Reglas de Integridad y son dos:

1. Integridad de entidad. «Ningún atributo que forme parte de la clave primaria puede tomar un valor nulo, entendiendo por tal, un valor desconocido o inaplicable». Restricción ya comentada al hablar del concepto de clave primaria.
2. Integridad referencial. «Si una tabla T2 tiene un atributo que es clave primaria de otra tabla T1, los valores de dicho atributo deben concordar con los de la clave primaria o tener valores nulos».

Este tipo de integridad se refiere al mantenimiento de referencias cruzadas entre las propias tablas o relaciones y para entender su significado es necesario introducir el concepto de «clave ajena».

«Dadas dos tablas o relaciones T1 y T2, se denomina clave ajena de la tabla T2 a un descriptor cuyos valores coinciden con los de la clave primaria de otra tabla T1». La clave ajena sirve para relacionar tablas.

Entonces, visto este concepto, la integridad referencial significa que los valores de la clave ajena de una tabla deben concordar con alguno de los valores que tome la clave primaria de la otra tabla.

Aclararemos todo esto con el siguiente ejemplo:

Ejemplo. Sean las siguientes tablas con sus respectivos atributos:

TABLA	ATRIBUTOS (LA CLAVE PRINCIPAL FIGURA EN NEGRITA)
T1: EMPLEADO	Cod-empl, Nombre, Provincia, Departamento, Cod-proy.
T2: PROYECTO	Cod-proy, Denominación-proy, Descripción-proy.

Si queremos saber datos concretos del proyecto (su denominación, etc.) en que trabaja un determinado empleado identificado por su código, tendremos que acudir a la tabla «Proyecto» utilizando el atributo Cod-proy de la tabla «Empleado» como referencia para encontrar la tupla asociada en la tabla «Proyecto». Por tanto, Cod-Proy es una clave referencial o clave ajena en la tabla «Empleado» y es clave principal o primaria en la tabla «Proyecto».

Trasladando el concepto de integridad referencial a este ejemplo, como T1 (relación que referencia) tiene un atributo (Cod-proy) que es clave primaria de T2 (relación referenciada), cualquier valor de dicho atributo en T1 debe concordar con un valor de la clave primaria (Cod-proy) de T2, o bien ser nulo, entendiendo por tal, una indicación de que es un valor desconocido, inaplicable, etc., por ejemplo, estar en blanco.

El mantenimiento de la integridad referencial supone la realización de alguna acción cuando se borra o modifica alguna tupla en la tabla referenciada (T2 «Proyecto» en nuestro ejemplo). Esta acción debe ser alguna de las siguientes:

- Impedir la operación de borrado o modificación en T2, así se asegura que una vez establecida no se puede romper la relación entre dos tuplas de ambas tablas.

- Transmitir en cascada la modificación, es decir, borrar o modificar las tuplas de T1 que hacen referencia a la que se acaba de borrar o modificar en T2.
- Poner a nulo, esto es, asignar el valor nulo al atributo que hace de clave referencial (clave ajena) para mantener la integridad.
- Poner valor por omisión o lanzar un procedimiento de usuario. En este caso cuando se altera el valor del atributo referenciado (Cod-proy en T2) se pone como valor de la clave ajena un valor por omisión o se ejecuta un trozo de código escrito por el usuario que establezca algún valor que sirva para mantener la integridad referencial.

#### 4.2.2. Restricciones explícitas o de usuario.

Las restricciones explícitas, también llamadas restricciones de usuario, son aquellas que introducen éstos, o los desarrolladores del esquema de datos, en el modelo relacional para aumentar su contenido semántico.

Algunos tipos de restricciones explícitas son:

- Restricciones que limitan el rango de valores posibles de un atributo en una tabla o relación.
- Restricciones que ligan los valores de atributos de una tabla con valores de atributos de otra o de la misma tabla.
- Restricciones que limitan los atributos que pueden ser comparados.
- Restricciones que determinan condiciones que deben verificarse después de un cambio en la base de datos.
- Restricciones que indican las condiciones que deben verificarse para que una operación pueda llevarse a cabo.

#### 4.3. OPERADORES ASOCIADOS AL MODELO RELACIONAL.

El modelo relacional admite dos «lenguajes relacionales» que constituyen sendos procedimientos de consulta de información residente en la base de datos:

- La utilización de un lenguaje procedimental (álgebra relacional), mediante el que se proporciona una secuencia de operaciones que genera la respuesta a la consulta deseada.
- La utilización de un lenguaje no procedimental (cálculo relacional), mediante el que se proporciona una descripción formal de la información deseada, pero sin especificar cómo obtenerla.

##### 4.3.1. Operadores asociados al Álgebra Relacional.

El Álgebra Relacional es un lenguaje procedimental mediante el que se puede realizar cualquier acceso a la base de datos. Matemáticamente es un álgebra completo y constituye un sistema cerrado de operaciones definidas sobre relaciones que consta de unos operadores y unos operandos. Los operandos son tablas o relaciones y los resultados de aplicarles los operadores son, asimismo, tablas o relaciones que pueden tomarse como operandos en sucesivas operaciones.

Existen cinco operadores fundamentales del Álgebra Relacional y otros que pueden ser definidos en términos de éstos. Los operadores fundamentales asociados al Álgebra Relacional son los siguientes:

### 1. Unión.

La unión de dos relaciones  $R1$  y  $R2$ , que se expresa como  $(R1 \cup R2)$ , es el conjunto formado por todas las tuplas de  $R1$  y todas las tuplas de  $R2$ . Este operador sólo se puede aplicar a relaciones compatibles, es decir, que tengan los mismos atributos, y de igual grado.

La unión permite la inserción de nuevas tuplas dentro de una relación existente, donde estas tuplas formarían uno de los operandos de la relación.

### 2. Diferencia.

La diferencia entre dos relaciones  $R1$  y  $R2$ , que se expresa como  $(R1 - R2)$ , es el conjunto formado por todas las tuplas de  $R1$  que no están en  $R2$ . Este operador sólo se puede aplicar a relaciones compatibles, es decir, que tengan los mismos atributos, y de igual grado.

La diferencia permite el borrado de tuplas de una relación.

### 3. Producto cartesiano.

El producto cartesiano de dos relaciones  $R1$  y  $R2$ , de grados « $m$ » y « $n$ » respectivamente, que se expresa como  $(R1 \times R2)$ , es el conjunto formado por todas las posibles tuplas de (« $m$ »+« $n$ ») elementos en las que los « $m$ » primeros son de  $R1$  y los « $n$ » restantes son de  $R2$ .

Es decir, el producto cartesiano produce una relación de salida desde dos relaciones de entrada de grados cualesquiera, tal que la salida tiene un grado igual a la suma de las entradas y las tuplas resultantes se obtienen yuxtaponiendo a cada tupla de la primera relación de entrada todas y cada una de las tuplas de la segunda.

### 4. Proyección.

La proyección actúa sobre una relación para producir otra, tal que la nueva relación es un subconjunto de la primera, determinado por una lista de proyección que especifica qué atributos de la relación antigua van a pasar a formar parte de la nueva. Se expresa como  $\Pi_x(R1)$ , donde « $x$ » indica los atributos de  $R1$  que van a pasar a formar parte de la nueva relación.

### 5. Selección.

La selección actúa sobre una relación para producir otra, tal que la nueva relación es un subconjunto de la primera, determinado por un filtro, esto es, por un criterio aplicado a las tuplas anteriores que determina qué tuplas pasan a formar parte de la nueva relación. Se expresa como  $\sigma_F(R1)$ , donde « $F$ » indica la fórmula que hace de filtro de la relación  $R1$ .

La fórmula es una función de comparación entre atributos del mismo dominio o entre atributos y constantes, y puede incluir: operandos que son nombres de atributos de la relación  $R1$ , operadores aritméticos de comparación (<, >, =, ...) u operadores lógicos (and  $\wedge$ , or  $\vee$ , not  $\neg$ ).

El siguiente ejemplo nos ayudará a comprender mejor el significado de estos operadores fundamentales del Álgebra Relacional.



Ejemplo. Dadas las siguientes tablas con sus respectivos atributos:

R1	
CÓDIGO	PROVINCIA
01	Álava
02	Albacete
03	Alicante
28	Madrid

R2	
CÓDIGO	PROVINCIA
01	Álava
08	Barcelona

R1 $\cup$ R2	
CÓDIGO	PROVINCIA
01	Álava
02	Albacete
03	Alicante
08	Barcelona
28	Madrid

R1	
CÓDIGO	PROVINCIA
01	Álava
02	Albacete
03	Alicante
28	Madrid

R2	
CÓDIGO	PROVINCIA
01	Álava
08	Barcelona

R1 - R2	
CÓDIGO	PROVINCIA
02	Albacete
03	Alicante
28	Madrid

R1	
CÓDIGO	PROVINCIA
01	Álava
02	Albacete
03	Alicante
28	Madrid

R2
CANTIDAD
100
500

R1 $\times$ R2		
CÓDIGO	PROVINCIA	CANTIDAD
01	Álava	100
01	Álava	500
02	Albacete	100
02	Albacete	500
03	Alicante	100
03	Alicante	500
28	Madrid	100
28	Madrid	500

R1	
CÓDIGO	PROVINCIA
01	Álava
02	Albacete
03	Alicante
28	Madrid

Proyección  
«x» = provincia

$\Pi_x(R1)$
PROVINCIA
Álava
Albacete
Alicante
Madrid

R1	
CÓDIGO	PROVINCIA
01	Álava
02	Albacete
03	Alicante
28	Madrid

Selección  
 «F» = (Código = «01») OR  
 (Provincia > «Alicante»)

$\sigma_F(R1)$	
CÓDIGO	PROVINCIA
01	Álava
28	Madrid

Los cinco operadores vistos forman un conjunto relacionalmente completo, es decir, permiten obtener cualquier subconjunto de datos contenidos en una base de datos. No obstante, no son los únicos. Se pueden definir, dentro del Álgebra Relacional, otra serie de operadores adicionales, obtenidos a partir de los cinco fundamentales. Estos operadores adicionales son los siguientes:

- Intersección.

La intersección de dos relaciones R1 y R2, que se expresa como  $(R1 \cap R2)$ , es el conjunto formado por todas las tuplas presentes tanto en R1 como en R2. Este operador sólo se puede aplicar a relaciones compatibles, es decir, que tengan los mismos atributos, y de igual grado.

- Cociente.

El cociente entre dos relaciones R1 y R2, de grados «m» y «n» respectivamente, donde «m» > «n», que se expresa como  $(R1/R2)$ , es otra relación constituida por el conjunto de tuplas «t» de grado  $(m - n)$ , tales que para toda tupla de R2, la tupla «t» está en R1. Es decir, la relación cociente estará definida sobre el conjunto de atributos de R1 que no pertenecen a R2 y constituida por las tuplas que al completarse con las de R2 permiten obtener la relación R1.

Ejemplo.

R1	
CÓDIGO	PROVINCIA
01	Álava
02	Albacete
03	Alicante
28	Madrid

R2	
CÓDIGO	PROVINCIA
01	Álava
08	Barcelona
28	Madrid

$R1 \cap R2$	
CÓDIGO	PROVINCIA
01	Álava
28	Madrid

R1	
CÓDIGO	PROVINCIA
01	Álava
02	Albacete
03	Alicante
28	Madrid

R2	
CÓDIGO	
01	
02	
03	
28	

R1/R2	
PROVINCIA	
Álava	
Albacete	
Alicante	
Madrid	

- Unión natural o Join Natural.

La Unión Natural, también llamada Join Natural, es uno de los operadores más utilizados a la hora de acceder a la información almacenada en la base de datos.

La Unión Natural es un operador compuesto por tres operadores básicos del Álgebra Relacional: producto cartesiano, selección y proyección, que produce una operación resultado a partir de dos operaciones de entrada, en las que se busca la coincidencia de valores entre atributos de igual nombre que se derivan del mismo dominio.

Dadas dos relaciones R1 y R2 definidas sobre conjuntos de atributos cuya intersección es no vacía, la unión natural de R1 y R2, que se expresa como  $R1 \times R2$ , se calcula de la siguiente manera:

1. Se calcula el producto cartesiano de  $R1 \times R2$ . Los atributos comunes a ambas relaciones se diferencian en  $R1 \times R2$  anotándolos como  $R1.A_i$  y  $R2.A_i$ .
2. Para cada atributo  $A_i$  que sea común a R1 y R2, se seleccionan las filas en las que el valor de  $R1.A_i$  coincide con el valor de  $R2.A_i$ .
3. Realizada la selección, se eliminan las columnas correspondientes a los atributos  $R2.A_i$ .

Ejemplo. Calcular la Unión Natural de las siguientes relaciones R1 y R2:

R1	A	B	C
	1	2	3
	4	5	6
	7	8	9

R2	B	C	D
	5	6	9
	3	3	8
	2	3	3

Primero. Calculamos  $R1 \times R2$ .

R1	A	B <sub>R1</sub>	C <sub>R1</sub>	B <sub>R2</sub>	C <sub>R2</sub>	D
	1	2	3	5	6	9
	1	2	3	3	3	8
	1	2	3	2	3	3
	4	5	6	5	6	9
	4	5	6	3	3	8
	4	5	6	2	3	3
	7	8	9	5	6	9
	7	8	9	3	3	8
	7	8	9	2	3	3

Segundo. Seleccionamos las tuplas donde los valores de los atributos  $B_{R1}$  y  $B_{R2}$  por un lado, y  $C_{R1}$  y  $C_{R2}$  por otro, coincidan. Es decir, hacemos  $\sigma_{B_{R1} = B_{R2} \wedge C_{R1} = C_{R2}} (R1 \times R2)$ .

$\sigma$	A	$B_{R1}$	$C_{R1}$	$B_{R2}$	$C_{R2}$	D
	1	2	3	2	3	3
	4	5	6	5	6	9

Tercero. Eliminamos los atributos  $B_{R2}$  y  $C_{R2}$  con lo que tendremos la Unión Natural  $R1 \times R2$ . Esto es:  $R1 \times R2 = \Pi_{A, B_{R1}, C_{R1}, D} [\sigma_{B_{R1} = B_{R2} \wedge C_{R1} = C_{R2}} (R1 \times R2)]$ .

$R1 \times R2$	A	B	C	D
	1	2	3	3
	4	5	6	9

#### 4.3.2. Ejemplos de acceso a bases de datos mediante el Álgebra Relacional.

Dada la importancia que tiene el Álgebra Relacional para acceder a la información de las bases de datos, veamos los siguientes ejemplos.

Sea una base de datos relacional constituida por las tres relaciones siguientes (la clave principal de cada relación figura subrayada):

PROVEEDOR (Cod-pro, Nombre-pro, Ciudad, Calle).

ARTÍCULO (Cod-art, Nombre-art, Precio, Descripción).

PEDIDO (Cod-pro, Cod-art, Cantidad).

ACCESO 1: OBTENER EL NOMBRE DE LOS PROVEEDORES QUE VIVEN EN MADRID.

El acceso consiste en seleccionar las tuplas de la relación PROVEEDOR donde la ciudad coincide con Madrid y proyectar sobre el atributo que proporciona el nombre del proveedor. Es decir:

$$\Pi_{\text{nombre-pro}} [\sigma_{\text{ciudad} = \text{Madrid}} (\text{PROVEEDOR})]$$

ACCESO 2: NOMBRE DE LOS PROVEEDORES QUE SUMINISTRAN EL ARTÍCULO DE CÓDIGO 15.

Los atributos involucrados en la petición (Cod-art y Nombre-pro) se encuentran en dos relaciones distintas. Por tanto, es necesario utilizar la Unión Natural para poder relacionar la información contenida en las tablas de PROVEEDOR y PEDIDO. Si bien al atributo Cod-art se puede obtener de las tablas de

ARTÍCULO y PEDIDO, se deberá utilizar esta última ya que es la que tiene algún atributo común con la tabla PROVEEDOR, y ésta era una condición imprescindible para poder realizar la Unión Natural.

El acceso pedido será:

$$\Pi_{\text{nombre-pro}} [\sigma_{\text{cod-art}=15} (\text{PEDIDO} * \text{PROVEEDOR})]$$

ACCESO 3. NOMBRE DE LOS PROVEEDORES QUE SUMINISTRAN ARTÍCULOS DE PRECIO MAYOR QUE 100.

Para realizar este acceso se deberá acceder a los atributos Precio de la tabla ARTÍCULO y Nombre-pro de la tabla PROVEEDOR, pero como estas tablas no tienen ningún atributo común, para poder realizar la Unión natural se deberá utilizar la tabla PEDIDO como eslabón intermedio, ya que las dos tablas anteriores comparten atributos con ella.

El acceso pedido será:

$$\Pi_{\text{nombre-pro}} [\sigma_{\text{precio} > 100} (\text{ARTÍCULO} * \text{PEDIDO} * \text{PROVEEDOR})]$$

#### 4.3.3. Operadores asociados al Cálculo Relacional.

El concepto de Cálculo Relacional fue propuesto por Codd como alternativo al Álgebra Relacional. A diferencia de esta última, en que han de especificarse los operadores que son necesarios para obtener el resultado deseado (lenguaje procedimental), con los lenguajes basados en el Cálculo Relacional se indica cuál es el resultado deseado, expresándolo mediante cálculo de predicados de primer orden (lenguaje predicativo).

Existen dos formas de Cálculo Relacional:

- a) Cálculo Relacional de Tuplas. Las variables representan tuplas de una relación. Un «query» se especifica mediante el conjunto de tuplas que satisfacen una fórmula bien formada (fbf).

Una «fórmula bien formada» (fbf) se construye combinando condiciones a cumplir por determinados elementos de las tuplas, operadores booleanos (and  $\wedge$ , or  $\vee$ , not  $\neg$ ) y cuantificadores (para todo  $\forall$ , existe  $\exists$ , ...).

- b) Cálculo Relacional de Dominios. Las variables representan valores de dominios sobre los que está definida la relación.

### 5. TEORÍA DE LA NORMALIZACIÓN.

Cuando se aborda el diseño de una base de datos relacional la primera tarea que debe realizarse es la determinación de los atributos de interés y de las dependencias funcionales existentes entre ellos. En otras palabras, lo primero es definir el esquema relacional  $R(T,L)$ , donde T representa el conjunto de atributos de la relación y L el conjunto de dependencias funcionales entre los atributos. Ahora bien, si se implementara sin más el esquema relacional obtenido, se presentarían una serie de problemas, denominados anomalías, que principalmente son los siguientes:

- Anomalías de repetición. Puede que cierta información esté repetida innecesariamente.
- Anomalías de actualización. Como consecuencia de las repeticiones, las actualizaciones pueden afectar a múltiples filas.
- Anomalías de inserción. Puede ser imposible añadir información a la base de datos.
- Anomalías de borrado. El borrado de una fila podría implicar pérdida de información.

Estos problemas se deben a que se combinan en una misma tabla atributos que, por las dependencias funcionales existentes entre ellos, no deberían aparecer juntos. Por tanto, para evitar estas anomalías, la solución es separar en tablas distintas los atributos que no deban aparecer juntos, es decir, descomponer el esquema relacional original  $R(T,L)$  en varios subesquemas. El proceso de sustitución de un esquema por otros se denomina normalización.

La normalización es una técnica de diseño de bases de datos relacionales que consiste en reemplazar un conjunto dado de relaciones por otro conjunto de relaciones que tengan una estructura más simple y más regular. La normalización elimina dependencias entre atributos y proporciona una estructura más regular en la representación de las tablas, constituyendo el soporte para el diseño de bases de datos relacionales.

El método más usado de normalizar consiste en descomponer, iterativamente, una relación en dos o más de mayor forma normal. Los requerimientos de la normalización exigen que no existan pérdidas de información ni dependencias.

Una tabla o relación está en una determinada forma normal si satisface un cierto conjunto de restricciones sobre los atributos. Las formas normales definen una serie de propiedades que deberá cumplir un esquema relacional (ya sea el original o algún subesquema), de forma que no se produzcan ninguna de las anomalías antes reseñadas.

El Modelo Relacional contempla cinco formas normales. Las tres primeras fueron definidas por Codd. Posteriormente Boyce y Codd definieron una versión modificada de la tercera forma normal, y más tarde se definieron la cuarta y quinta forma normal. Nosotros sólo estudiaremos las tres primeras, que son las que se refieren a las dependencias funcionales. Para ello, comenzaremos por definir los conceptos relativos a la idea de «dependencia».

### 5.1. DEPENDENCIA FUNCIONAL ENTRE ATRIBUTOS.

Las dependencias entre los atributos, junto con las restricciones semánticas, conforman los elementos semánticos del modelo relacional. Las dependencias funcionales son el elemento semántico más importante. Los conceptos fundamentales que hacen referencia a las mismas son los siguientes:

- Dependencia funcional. Se dice que un descriptor o conjunto no vacío de atributos  $B$  depende funcionalmente de otro descriptor  $A$  de una misma tabla o relación, si y sólo si a cada valor de  $A$  le corresponde un único valor de  $B$ .

Se representa como:  $A \rightarrow B$  y se lee como « $B$  depende de  $A$ ».

La dependencia funcional quiere decir que independientemente de las tuplas existentes en la tabla en un momento dado, el valor del atributo A determina unívocamente el del atributo B.

Por ejemplo: dada la tabla EMPLEADO (Cod-empl, Salario, Categoría, Proyecto, ---), los atributos Salario y Categoría dependen funcionalmente de Cod-empl, puesto que para un valor de Cod-empl, sólo le corresponde un valor de Salario y un valor de Categoría. Sin embargo, el atributo Proyecto no depende funcionalmente de Cod-empl porque para un valor de este atributo, Proyecto puede tomar varios valores (todos los correspondientes a los proyectos en que trabaje el empleado).

- Dependencia funcional completa. Se dice que un atributo B tiene dependencia funcional completa de un conjunto de atributos A de una misma tabla o relación, si B depende funcionalmente de A, pero no de ningún subconjunto obtenido de los posibles atributos que formen A.

Se representa como:  $A \twoheadrightarrow B$

Por ejemplo: Dada la tabla ASIGNACIÓN (Cod-empl, Cod-proy, Horas trabajadas en el proyecto, ---), el atributo Horas trabajadas en el proyecto tiene dependencia funcional completa de la clave ya que para un empleado y un proyecto determinado, el valor que toma el número de horas trabajadas es único y, sin embargo, este atributo no depende ni de Cod-empl ni de Cod-proy, subconjuntos de la clave.

- Dependencia transitiva. Dados los atributos (o conjunto de atributos) A, B y C de una tabla o relación, si se cumple que B depende funcionalmente de A y que C depende funcionalmente de B, pero A no depende funcionalmente de B, entonces se dice que C depende transitivamente de A.

Expresado con otras palabras, se dice que un descriptor depende transitivamente de otro, si y sólo si depende de él a través de otro descriptor.

Por ejemplo: Dada la tabla EMPLEADO (Cod-empl, Salario, Categoría, Proyecto, ---), el atributo Salario depende de Categoría y éste, a su vez, depende de Cod-empl, y además, Categoría no implica Cod-empl, por tanto, Salario depende transitivamente de Cod-empl a través de Categoría.

- Axiomas de Armstrong. Se conoce por este nombre a un conjunto de reglas que permiten deducir unas dependencias a partir de otras. Son las siguientes:

– Axioma de reflexividad: todo descriptor depende funcionalmente de sí mismo.

$$\forall A, A \twoheadrightarrow A$$

- Axioma de aumentatividad: si un descriptor B depende funcionalmente de otro descriptor A de la misma tabla, también depende funcionalmente de cualquier descriptor A' del que A sea un subconjunto.

$$\text{Si } A \twoheadrightarrow B, \text{ entonces } A' \twoheadrightarrow B, \forall A' \supseteq A$$

- Axioma de proyectividad: si un descriptor B depende funcionalmente de otro descriptor A de la misma tabla, cualquier subconjunto B' del descriptor B también depende funcionalmente de A.

si  $A \rightarrow B$ , entonces  $A \rightarrow B'$ ,  $\forall B' \subseteq B$

- Axioma de aditividad: si  $A \rightarrow B$  y  $C \rightarrow D$ , entonces  $A \cup C \rightarrow B \cup D$
- Axioma de transitividad: si  $A \rightarrow B$  y  $B \rightarrow C$ , entonces  $A \rightarrow C$

Vistos los conceptos de dependencia, vamos a definir las formas normales.

## 5.2. PRIMERA FORMA NORMAL.

Una tabla, relación o esquema relacional R está en Primera Forma Normal 1FN, si no contiene grupos repetitivos, también llamados atributos múltiples. Es decir, una tabla está en 1FN si todos los atributos no principales dependen funcionalmente de la clave.

Por ejemplo, sea la tabla LIBRO (Cod-libro, Título, Nombre-autores).

LIBRO	COD-LIBRO	TÍTULO	NOMBRE-AUTORES
	100100	Cálculo Matemático	Luis García, Vicente Arranz
	100101	Cálculo Infinitesimal	Eduardo Villar
	200100	Redes de Ordenadores	Juan Riera, Juan Viñas
	300105	Contabilidad Analítica	José Rivero

Esta tabla no está en 1FN ya que al poder estar firmado un libro por varios autores, el atributo Nombre-autores no depende funcionalmente de la clave Cod-libro. Para que estuviera en 1FN, la tabla se debería sustituir por otra donde el atributo Nombre-autor pasaría a formar parte de la clave y se aumentaría el número de tuplas.

LIBRO	COD-LIBRO	TÍTULO	NOMBRE-AUTORES
	100100	Cálculo Matemático	Luis García
	100100	Cálculo Matemático	Vicente Arranz
	100101	Cálculo Infinitesimal	Eduardo Villar
	200100	Redes de Ordenadores	Juan Riera
	200100	Redes de Ordenadores	Juan Viñas
	300105	Contabilidad Analítica	José Rivero

Esta nueva tabla, donde la clave es Cod-libro, Nombre-autor, sí que está en 1FN.



### 5.3. SEGUNDA FORMA NORMAL.

Una tabla, relación o esquema relacional está en Segunda Forma Normal 2FN, si está en 1FN y si cada atributo no principal tiene una dependencia funcional completa de la clave. Dicho de otra forma, una tabla está en 2FN, si cada atributo no principal de la tabla depende de toda la clave.

Por ejemplo, sea la tabla anterior a la que hemos añadido el atributo Cod-autor como parte de la clave, pasando al atributo Nombre-autor a ser un atributo no principal.

LIBRO	COD-LIBRO	COD-AUTOR	TÍTULO	NOMBRE-AUTOR
	100100	45	Cálculo Matemático	Luis García
	100100	63	Cálculo Matemático	Vicente Arranz
	100101	27	Cálculo Infinitesimal	Eduardo Villar
	200100	52	Redes de Ordenadores	Juan Riera
	200100	70	Redes de Ordenadores	Juan Viñas
	300105	15	Contabilidad Analítica	José Rivero

Esta tabla está en 1FN, puesto que no contiene grupos repetitivos, pero no está en 2FN, ya que el atributo no principal, Nombre-autor, no depende de toda la clave Cod-libro, Cod-autor y sin embargo depende de un subconjunto de ésta, en concreto, de Cod-autor.

Para poner la tabla anterior en 2FN, descompondríamos ésta en dos subesquemas:

- R1 (Cod-libro, Cod-autor, Título) que guarda la información relativa a Títulos.
- R2 (Cod-autor, Nombre-autor) que guarda la información relativa a Autores.

### 5.4. TERCERA FORMA NORMAL.

Una tabla, relación o esquema relacional está en Tercera Forma Normal 3FN, si está en 2FN y si cada atributo no principal no depende transitivamente de la clave. En otras palabras, la tabla está en 3FN, si cada uno de los atributos no principales de la tabla dependen única y exclusivamente de la clave.

La definición de 3FN también se puede expresar de la siguiente manera: «Una tabla, relación o esquema  $R(T,L)$  está en 3FN, si está en 2FN y si para toda dependencia funcional  $X \rightarrow A$  no trivial (es decir, si  $A \notin X$ ) se cumple al menos una de las siguientes condiciones:

- $X$  es clave de la tabla, relación o esquema  $R(T,L)$ .
- $A$  es un atributo que pertenece a alguna clave».

Por ejemplo, sea la tabla EMPLEADO (DNI-Epleado, Nombre, Provincia-nacimiento, Capital-provincia).

Las dependencias funcionales existentes en esta relación son:

- DNI  $\rightarrow$  Nombre, Provincia, Capital
- Provincia  $\rightarrow$  Capital

Esta última dependencia no cumple ninguna de las dos condiciones de la 3FN, puesto que ni Provincia es clave, ni Capital es un atributo principal. Para que la relación estuviera en 3FN, habría que descomponerla en dos subesquemas:

- R1 (DNI-empleado, Nombre, Provincia-nacimiento)
- R2 (Provincia-nacimiento, Capital-provincia)

## 5.5. OTRAS FORMAS NORMALES.

- Forma Normal de Boyce-Codd.

Una tabla, relación o esquema relacional  $R(T,L)$  está en la Forma Normal de Boyce-Codd FNBC, si está en 3FN y si para toda dependencia funcional  $X \rightarrow A$  no trivial se cumple que  $X$  es clave de la tabla o esquema  $R(T,L)$ . Expresada de otra manera esta definición, una tabla está en FNBC si y sólo si está en 3FN y todo determinante es clave candidata.

- Cuarta Forma Normal.

Una tabla, relación o esquema relacional  $R(T,L)$  está en Cuarta Forma Normal 4FN, si y sólo si está en FNBC y en todas las dependencias múltiplemente valoradas, el implicante es clave candidata.

Una dependencia múltiplemente valorada es un caso especial de dependencia en el que el valor del implicante determina un conjunto bien definido de posibles valores para el implicado.

- Quinta Forma Normal.

Una tabla, relación o esquema relacional  $R(T,L)$  está en Quinta Forma Normal 5FN, si y sólo si está en 4FN y toda dependencia de combinación está implicada por una clave candidata.

Una dependencia de combinación aparece en relaciones que no pueden descomponerse en otras sin perder información. Para evitar esa pérdida de información hay que descomponerlas, al menos, en otras tres relaciones.

Las formas normales que se han estudiado definen propiedades que debe cumplir cada esquema relacional. Cada vez que se detecta que un esquema no cumple la Forma Normal deseada, se debe llevar a cabo una descomposición. Para que ésta se haga con garantías, de manera que no se pierda información por el camino, se tratarán de cumplir las propiedades de «unión sin pérdida de información» o propiedad LJ (del inglés «Losless Join») y la de «preservación de las dependencias».

Dado un esquema  $R(T, L)$  y la descomposición en subesquemas  $\rho = (R_1, R_2, \dots, R_n)$ , se dice que  $\rho$  es una descomposición que cumple la propiedad LJ respecto de  $R$  si para toda ocurrencia « $r$ » de  $R$  se cumple que:

$$r = \Pi_{T_1}(r) * \Pi_{T_2}(r) * \dots * \Pi_{T_n}(r)$$

Para descomponer un esquema  $R(T, L)$  en un conjunto de subesquemas que cumplan la FNBC y la propiedad LJ, se aplica el Teorema de Dolobell-Casey:

«Dado un esquema  $R(T, L)$  donde el conjunto de dependencias  $L$  es de la forma  $L = \{X \rightarrow Y \text{ tal que } (X \cup Y) \subseteq T\}$ , la descomposición de  $R$  en dos subesquemas  $R_1(T_1, L_1)$  y  $R_2(T_2, L_2)$  verifica la propiedad LJ si y sólo si se cumple al menos alguna de las siguientes dependencias funcionales en  $L$ :

$$(T_1 \cap T_2) \rightarrow (T_1 - T_2) \text{ o}$$

$$(T_1 \cap T_2) \rightarrow (T_2 - T_1)»$$

## 6. REPRESENTACIÓN DEL MODELO LÓGICO DE DATOS. EL DIAGRAMA DE ESTRUCTURA DE DATOS.

El Diagrama de Estructura de Datos (DED) es una técnica de representación gráfica del modelo lógico de datos.

El DED permite representar esquemas tanto relacionales como jerárquicos o Codasy1, si bien, dado que como modelo lógico estamos considerando el relacional, será para éste esquema para el que se utilice la representación vía DED.

En un DED las entidades o tablas se representan mediante un rectángulo con el nombre de la entidad dentro, y las interrelaciones entre entidades, mediante una línea recta que las une. Esta línea puede acabar en un tridente para indicar una cardinalidad de la interrelación superior a 1.

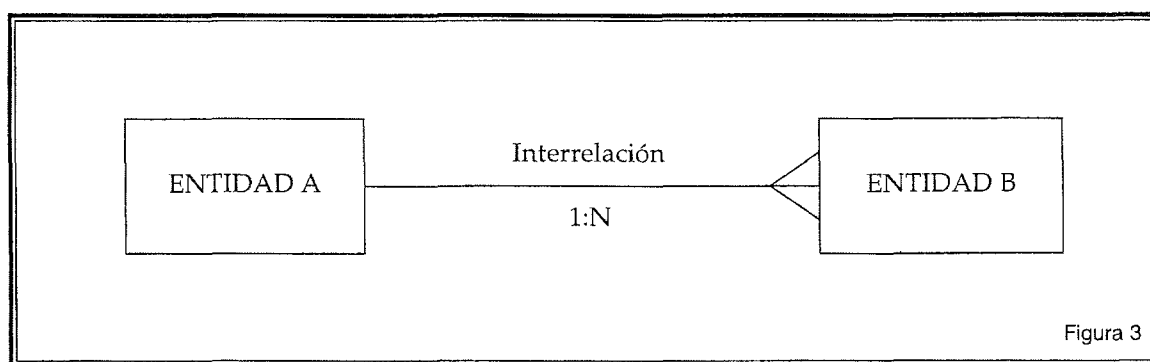


Figura 3

Las reglas de representación gráfica que sigue el DED son las siguientes:

1. Las interrelaciones entre entidades serán siempre binarias o de grado dos. En el caso de que sean de mayor grado, habrá que descomponerlas en interrelaciones binarias.

2. Sólo se consideran interrelaciones de cardinalidad 1:N. Para otro tipo de interrelaciones se procederá del siguiente modo:

- a) En el caso de cardinalidad 1:1 se agruparán las dos entidades en una sola, añadiéndose los atributos de una entidad a la otra. Otra posibilidad, menos usual, es conservar las dos entidades estableciendo una interrelación entre ambas que puede ir en cualquiera de los dos sentidos.
- b) En el caso de cardinalidad M:N se creará una entidad auxiliar que sirva de enlace o nexo entre las dos entidades iniciales, obteniéndose así dos interrelaciones 1:N. La clave de esta entidad de enlace será una clave compuesta, formada por la concatenación de las claves de cada una de las entidades originales.

3. Dada una interrelación 1:N entre dos entidades A y B, se representarán las interrelaciones obligatorias y opcionales de la siguiente manera:

- a) Si para toda ocurrencia de A debe existir siempre al menos una ocurrencia de B asociada y para una ocurrencia de B existe una de A asociada, la interrelación es obligatoria en ambos extremos.

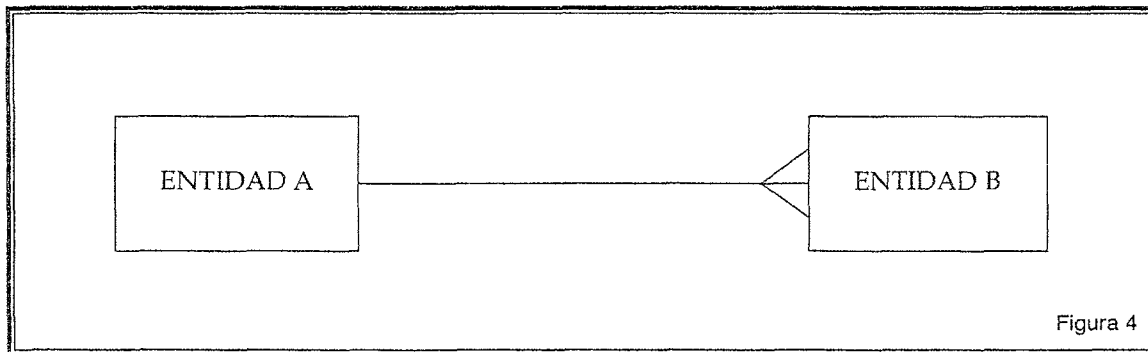


Figura 4

- b) Si para toda ocurrencia de A pueden existir, o no, una o varias ocurrencias de B asociadas, y para una ocurrencia de B siempre existe una de A asociada, la interrelación es opcional en A y obligatoria en B.

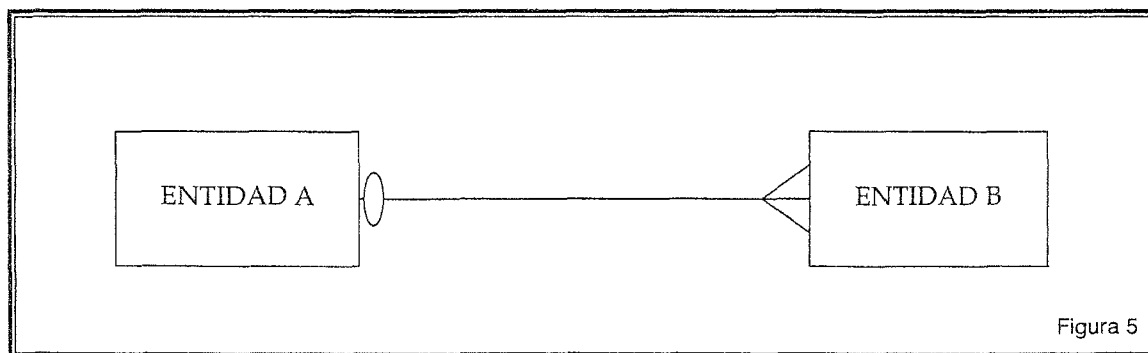
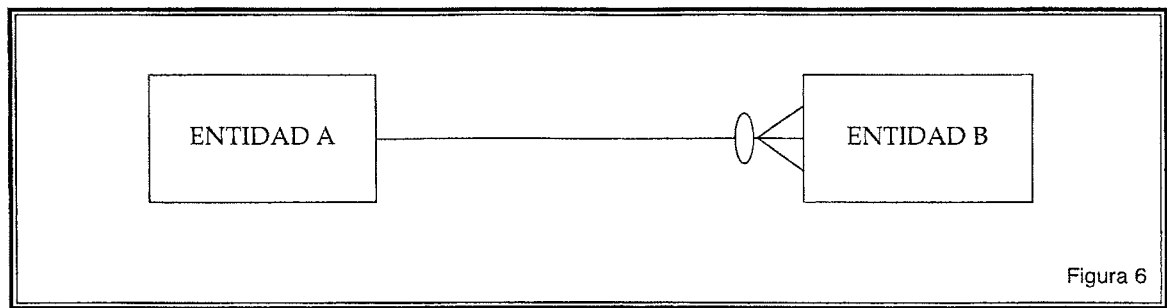
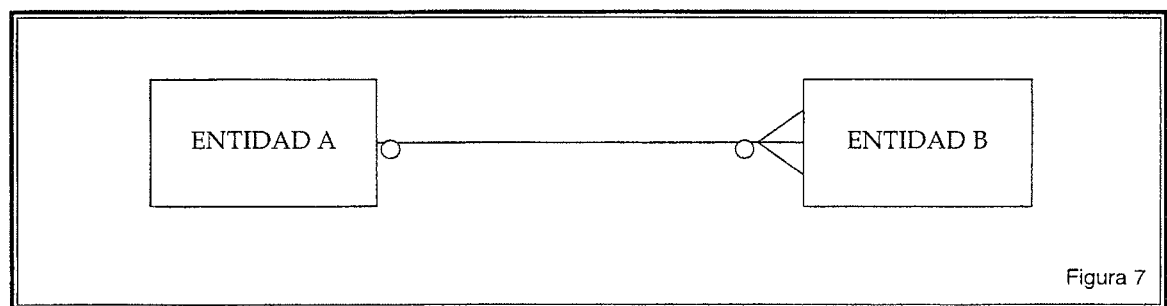


Figura 5

- c) Si para una ocurrencia de A debe existir siempre al menos una ocurrencia de B asociada y para una ocurrencia de B puede existir, o no, una de A asociada, la interrelación es obligatoria en A y opcional en B.



- d) Finalmente, si para una ocurrencia de A puede existir, o no, una ocurrencia de B asociada y para una ocurrencia de B puede existir, o no, una de A asociada, la interrelación es opcional en ambos extremos.



## 6.1. TRANSFORMACIÓN DEL MODELO CONCEPTUAL A UN MODELO RELACIONAL.

Dado un sistema de información, plasmar el universo del discurso, es decir, aquella parte del mundo real relevante para dicho sistema, es tanto como obtener su modelo de datos. Para ello los pasos que se habrán de seguir son los siguientes:

- A) Obtención del modelo conceptual de datos a través de la técnica del Modelo E/R.
- B) Obtención del modelo lógico de datos (esquema relacional) mediante una serie de reglas de transformación que permiten pasar del modelo conceptual al lógico, utilizando también para ello la teoría de la normalización.

En concreto, las reglas de transformación que se aplican para pasar del modelo E/R a un esquema relacional son las siguientes:

1. Toda entidad representada en el modelo E/R se convierte en una tabla o relación que toma el nombre de la entidad. Los atributos de la entidad son las columnas de la tabla y el atributo identificador principal es la clave primaria.
2. Las interrelaciones M:N del modelo E/R se transforman en una tabla cuya clave primaria estará compuesta por las claves de las entidades que asocia.
3. Las interrelaciones 1:N del modelo E/R se transforman propagando el atributo identificador principal de la entidad que tiene cardinalidad máxima 1 (entidad maestra) a la que tiene cardinalidad máxima N (entidad de detalle). El atributo propagado es una clave ajena que referencia a la tabla con cardinalidad máxima 1.

4. Las interrelaciones 1:1 del modelo E/R se transforman propagando la clave y los demás atributos en cualquier sentido.
5. Finalmente, se normalizarán hasta la 3FN las tablas o relaciones obtenidas y se representará el esquema relacional obtenido mediante un DED.

Otra posibilidad es obtener directamente el esquema relacional a partir del universo del discurso, sin haber elaborado previamente el esquema conceptual, las etapas para su construcción son las siguientes:

1. Identificar y representar las entidades dentro del sistema.
2. Determinar las claves o identificadores de las entidades.
3. Establecer y representar las interrelaciones entre las entidades, analizar la cardinalidad y el grado de las interrelaciones encontradas e indicar la obligatoriedad o la opcionalidad de la interrelación.
4. Identificar y describir los atributos de cada entidad.
5. Verificar el esquema obtenido. Para ello, se eliminarán las posibles redundancias en las interrelaciones. Una interrelación es redundante si se puede expresar exactamente por medio de una combinación de otras interrelaciones.

## 6.2. EJEMPLO PRÁCTICO.

Obtener el modelo lógico de datos en tercera forma normal de un sistema de gestión de albaranes cuyos datos, identificados inicialmente como la entidad «ALBARÁN», son los siguientes:

- Código del pedido (clave).
- Fecha del pedido.
- Código del proveedor.
- Nombre del proveedor.
- Dirección del proveedor.
- Código del material.
- Descripción del material.
- Cantidad pedida del material.
- Precio unitario del material.

### SOLUCIÓN

PRIMERA FORMA NORMAL:

No se cumple ya que para un mismo código de pedido hay varios códigos de material. Es decir, código de material no depende funcionalmente del código de pedido.

Para lograr la 1FN, una vez identificados los atributos que no dependen funcionalmente de la clave, se forma con ellos una nueva entidad, eliminándolos de la antigua. La clave de la nueva entidad estará compuesta por uno o varios de sus atributos más la clave de la antigua.

En este ejemplo, la entidad «ALBARÁN» se transforma en las entidades:

**PEDIDO:**

- Código del pedido.
- Fecha del pedido.
- Código del proveedor.
- Nombre del proveedor.
- Dirección del proveedor.

**PEDIDO/MATERIAL:**

- Código del pedido.
- Código del material.
- Descripción del material.
- Cantidad pedida del material.
- Precio unitario del material.

**SEGUNDA FORMA NORMAL:**

En la entidad «PEDIDO/MATERIAL» los atributos «Descripción del material» y «Precio unitario del material» no tienen una dependencia funcional completa de la clave, sino que la tienen sólo de una parte de la misma (Código del material).

Para poner la entidad en 2FN, una vez identificados los atributos que no dependen funcionalmente de toda la clave sino de parte de la misma, se forma con ellos una nueva entidad y se quitan de la antigua. La clave de esta nueva entidad será una parte de la antigua de la que dependen funcionalmente.

En este ejemplo, la entidad «Pedido/Material» se transforma en las entidades:

**MATERIAL:**

- Código del material.
- Descripción del material.
- Precio unitario del material.

**PEDIDO/MATERIAL:**

- Código del pedido.
- Código del material.
- Cantidad pedida del material.

**TERCERA FORMA NORMAL:**

Teniendo en cuenta la definición de 3FN, en la entidad «PEDIDO», se observa la siguiente dependencia transitiva:

- Para un Código del pedido (A) hay un único Código del proveedor (B), por tanto, B depende funcionalmente de A.
- Para un Código del proveedor (B) hay un único Nombre y Dirección del proveedor (C), por tanto, C depende funcionalmente de B.
- Ahora bien, para un Código del proveedor (B) hay varios Códigos del pedido (A), por tanto, C depende transitivamente de A, y consecuentemente, la entidad no está en 3FN.

Para ponerla en 3FN, una vez identificados los atributos que dependen de otro atributo distinto de la clave (las dependencias transitivas), se formará con ellos una nueva entidad y se quitarán de la antigua. La clave de la nueva entidad será el atributo del que dependen, Este atributo pasará a ser una clave ajena en la entidad antigua.

En el ejemplo, la entidad «pedido» se transforma en las siguientes entidades:

**PROVEEDOR:**

- Código del proveedor.
- Nombre del proveedor.
- Dirección del proveedor.

**PEDIDO 2:**

- Código del pedido.
- Fecha del pedido.
- Código del proveedor.

Por tanto, de la entidad «ALBARÁN» se obtiene el siguiente modelo lógico de datos en 3FN:

ENTIDADES	ATRIBUTOS
PEDIDO:	<ul style="list-style-type: none"> <li>• Código del pedido</li> <li>• Fecha del pedido</li> <li>• Código del proveedor</li> </ul>
PROVEEDOR:	<ul style="list-style-type: none"> <li>• Código del proveedor</li> <li>• Nombre del proveedor</li> <li>• Dirección del proveedor</li> </ul>
MATERIAL:	<ul style="list-style-type: none"> <li>• Código del material</li> <li>• Descripción del material</li> <li>• Precio unitario del material</li> </ul>
PEDIDO / MATERIAL:	<ul style="list-style-type: none"> <li>• Código del pedido</li> <li>• Código del material</li> <li>• Cantidad pedida del material</li> </ul>

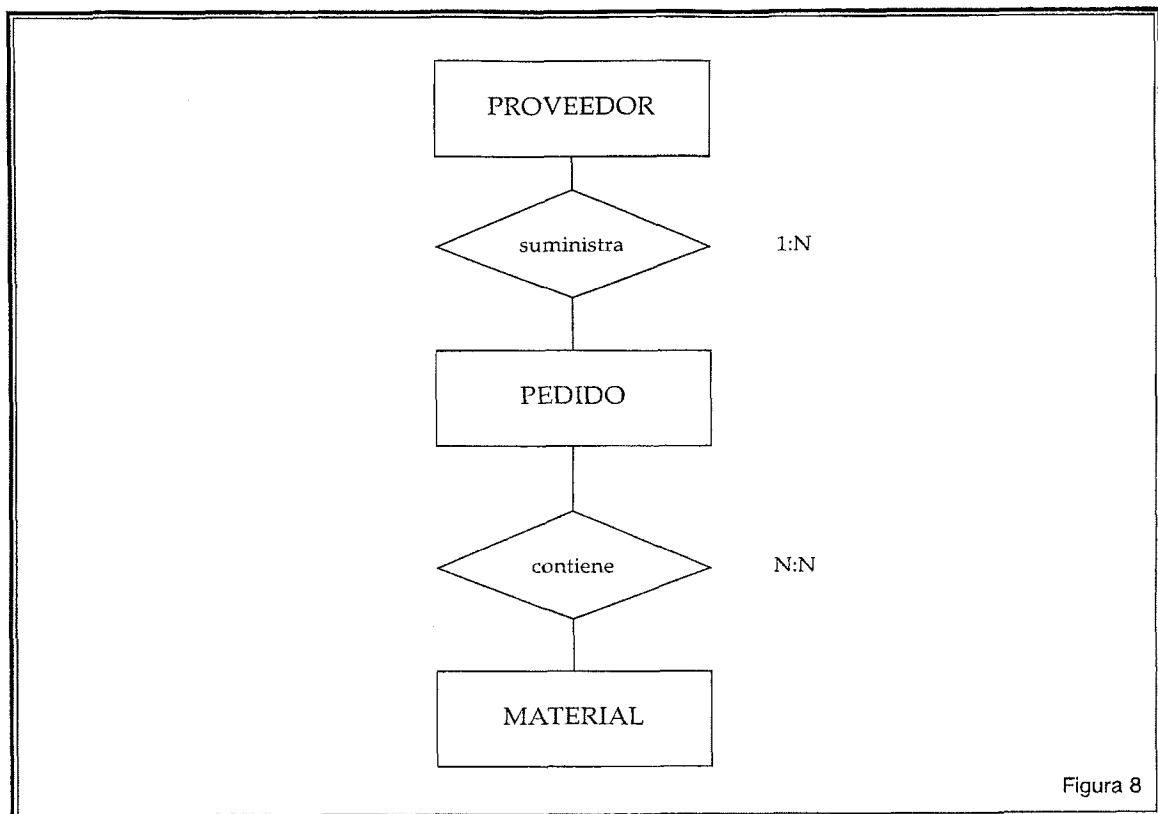
Vamos a realizar el mismo ejemplo partiendo del modelo conceptual.

Se trata de obtener el modelo lógico de datos en 3FN de un sistema de gestión de pedidos teniendo en cuenta las siguientes consideraciones: que cada pedido es suministrado por un sólo proveedor, el cual puede suministrar varios pedidos; y que cada pedido consta de varios materiales y cada material puede estar incluido en varios pedidos.

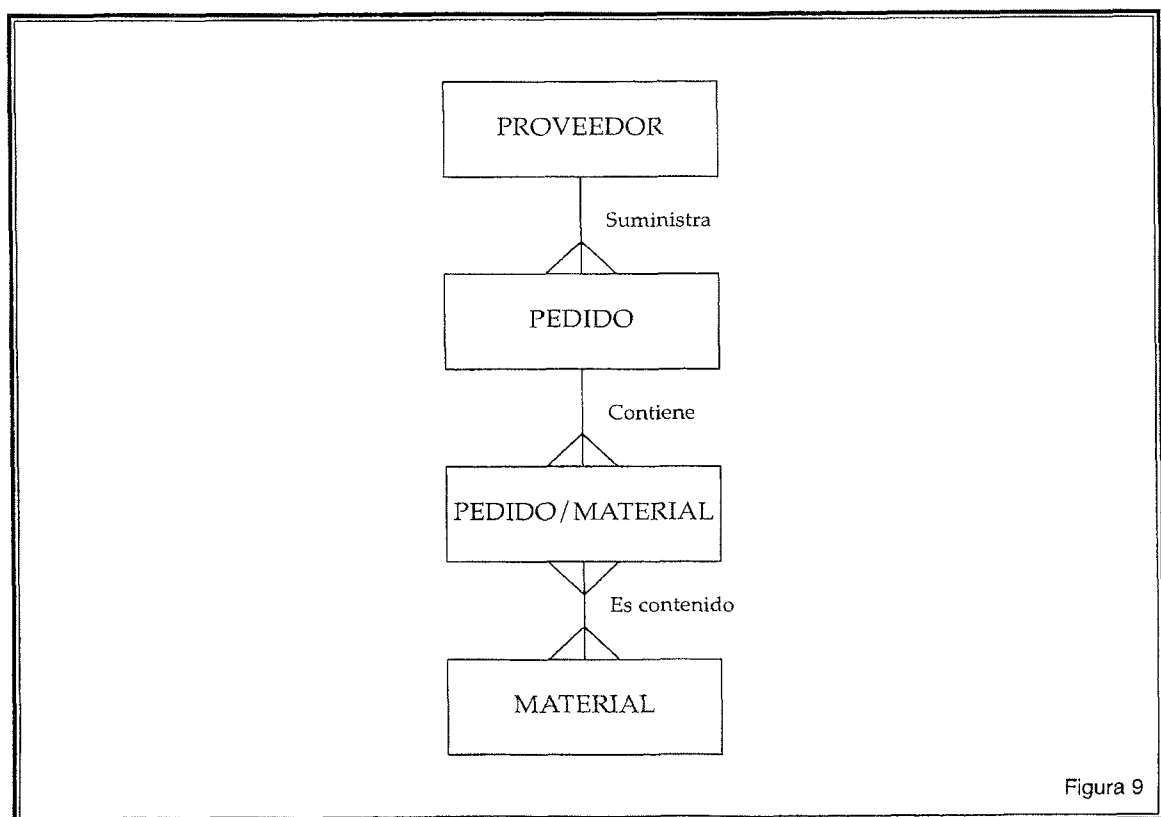
**SOLUCIÓN**

Modelo conceptual (Modelo Entidad /Relación de Chen):





Modelo lógico (Diagrama de estructura de datos):



Para llegar a la 3FN se tendrá en cuenta que:

Como la relación entre proveedor y pedido es 1:N, en la entidad maestra proveedor (cardinalidad 1) figurarán sus datos propios y en la entidad de detalle pedido (cardinalidad N) figurarán, además de sus datos propios, el Código de proveedor como clave ajena.

Como la relación entre pedido y material es M:N, en la entidad material figurarán sus datos propios y se crea una entidad de enlace pedido/material cuya clave estará compuesta por la de las entidades pedido y material.

Es decir, los atributos de cada entidad quedan de la siguiente forma:

ENTIDADES	ATRIBUTOS
PEDIDO:	<ul style="list-style-type: none"><li>• Código del pedido</li><li>• Fecha del pedido</li><li>• Código del proveedor</li></ul>
PROVEEDOR:	<ul style="list-style-type: none"><li>• Código del proveedor</li><li>• Nombre del proveedor</li><li>• Dirección del proveedor</li></ul>
MATERIAL:	<ul style="list-style-type: none"><li>• Código del material</li><li>• Descripción del material</li><li>• Precio unitario del material</li></ul>
PEDIDO / MATERIAL:	<ul style="list-style-type: none"><li>• Código del pedido</li><li>• Código del material</li><li>• Cantidad pedida del material</li></ul>

## 7. DISEÑO LÓGICO.

El diseño lógico, tal y como se ha comentado en el punto anterior, es la etapa de creación de la base de datos en la que se traduce el modelo conceptual obtenido en la etapa del diseño conceptual en modelo lógico y un esquema lógico expresado de un modo comprensible para un SGBD. Para los sistemas SGBD relacionales, el esquema lógico estará expresado en tablas y columnas (o relaciones y atributos).

El diseño lógico se puede dividir en dos etapas:

- **Diseño lógico estándar:** en esta etapa se obtiene un modelo lógico estándar y un esquema lógico estándar, independientes del SGBD en el que se vaya a implementar la base de datos. El modelo lógico estándar puede expresarse empleando varias técnicas, entre las que cabe citar el Diagrama de Estructura de Datos (DED) y el modelo relacional. El esquema lógico estándar se obtiene utilizando un Lenguaje de Definición de Datos (DDL) estándar, habitualmente el SQL-92 (que es el estándar ISO) y es el que utilizaremos en los ejemplos de este tema.
- **Diseño lógico específico:** utilizando el esquema lógico estándar que se ha obtenido, se estudia su implementación en un determinado SGBD (SQL Server, Oracle, DB2, Sybase, etc.). Para ello, habrá de analizarse la compatibilidad del modelo lógico estándar con el modelo lógico específico del SGBD elegido, y proponer un modo de solucionar aquellos aspectos del modelo lógico estándar que no recoge el modelo lógico específico. Una vez realizada esa tarea, se obtiene un esquema lógico específico usando el Lenguaje de Definición Datos propio del SGBD (normalmente, será un lenguaje SQL con algunas peculiaridades propias de cada SGBD). A esta etapa también se la conoce como etapa de implementación de la BD.

## 7.1. DISEÑO LÓGICO ESTÁNDAR.

El esquema lógico estándar será la expresión del modelo lógico estándar utilizando un Lenguaje de Definición de Datos (DDL) independiente del SGBD (SQL-2 o SQL.-3 habitualmente).

Las técnicas de modelado conceptual son diferentes de las técnicas de modelado lógico, por lo que habrá que convertir cada elemento presente en el modelo conceptual a los elementos existentes y expresables en la técnica de modelado lógico que hayamos seleccionado.

En las BD relacionales, es habitual usar el modelo E/R para el modelado conceptual y el modelo relacional para el modelado lógico. En este apartado vamos a estudiar la conversión de los elementos del E/R al modelo relacional.

### 7.1.1. Transformación de dominios.

Un dominio del modelo E/R se transforma en un dominio en el modelo relacional utilizando la sentencia SQL CREATE DOMAIN.

### 7.1.2. Transformación de entidades.

Cada entidad del modelo E/R se transformará en una tabla (relación) en el modelo relacional. La tabla tendrá el mismo nombre que la entidad de la que proviene. Vemos un ejemplo a continuación:

Para obtener el esquema lógico estándar, la tabla se definirá usando la sentencia CREATE TABLE.

### 7.1.3. Transformación de atributos.

Cada atributo de una entidad se transformará en una columna de la tabla (relación) del modelo relacional.

Los atributos de las entidades pueden ser de cuatro tipos:

- Atributos identificadores: se transforman en una columna que es la clave primaria de la tabla. En SQL, la condición de clave primaria se representará colocando la cláusula **PRIMARY KEY** al lado del nombre de la columna en la que se ha convertido el atributo (dentro de la sentencia **CREATE TABLE** con la que se crea la tabla en la que esta incluida la columna).
- Atributos identificadores alternativos: se transforman en una columna a la que se le añade la restricción **UNIQUE**, lo cual significa que no puede haber valores repetidos en esa columna.
- Atributos no identificadores: se transforman en una columna de la tabla.
- Atributos multivaluados: en el modelo relacional una instancia de una relación sólo toma un valor para cada atributo. Por ello, será obligatorio crear una nueva tabla que contenga a la clave primaria de la tabla anterior y al atributo multivaluado, siendo la clave primaria de la nueva tabla la concatenación de los dos atributos y marcándose la clave primaria de la tabla anterior como clave foránea en la nueva tabla.

#### **7.1.4. Transformación de interrelaciones N a M, 1 a N y 1 a 1.**

En el modelo relacional, la única unidad de modelado son las tablas, por lo que las interrelaciones del modelo E/R deben transformarse en tablas del modelo relacional produciéndose una pérdida de semántica.

El modo de transformar las interrelaciones en tablas depende del tipo de interrelación considerada. Los tipos de transformaciones existentes son:

A) Para interrelaciones del tipo N:M.

Una interrelación N:M se va a convertir en una nueva tabla del modelo relacional. La nueva tabla tendrá como columnas la concatenación de los atributos identificadores de las entidades que estaban unidas por la interrelación.

Si la interrelación tiene atributos en el modelo E/R, cada atributo de la interrelación pasará a ser una columna de la nueva tabla.

Los atributos identificadores de las entidades unidas mediante la interrelación serán claves primarias en las tablas del modelo relacional que representen a sus entidades. Por tanto, esos mismos atributos serán considerados claves foráneas en la tabla que representa a la interrelación.

La condición de clave ajena de una columna se expresará mediante la cláusula **FOREIGN KEY** en SQL-92.

B) Las relaciones 1:N se pueden transformar de dos maneras:

- No crear ninguna tabla que represente a la interrelación y añadir a la tabla que representa a la entidad con cardinalidad n el conjunto de atributos que son clave primaria de la entidad con cardinalidad 1. Éste es el modo habitual de realizar la transformación.

- Convirtiendo la interrelación en una tabla, siendo la clave primaria de la tabla el conjunto de atributos identificadores del lado n de la relación. Esta opción se utiliza en los siguientes casos:
  - Cuando se cree que en el futuro la relación se va a transformar en una de tipo N:M (y por tanto, será necesario tener una tabla que represente a esa relación).
  - Cuando la interrelación tiene atributos en el modelo relacional.
  - Cuando la interrelación es optativa para las ocurrencias de las entidades situadas en el lado 1 de la relación (cardinalidad 0:1) y el porcentaje de ocurrencias interrelacionadas es bajo, lo cual va a significar que en las columnas absorbidas en la tabla n van a existir muchos valores nulos.

#### C) Para interrelaciones del tipo 1:1.

Se realizan del mismo modo que las interrelaciones 1:N, pero teniendo en cuenta que si se decide no crear una tabla que represente a la interrelación, la elección de la tabla a la que se le añaden los atributos del otro extremo es optativa, si bien se suele seguir el siguiente criterio:

- Si una de las dos entidades de la interrelación tiene cardinalidad (0,1) y la otra entidad tiene cardinalidad (1,1), entonces se propagan los atributos identificadores de la tabla (1,1) a la tabla (0,1), evitándose los valores nulos.
- Si las dos tablas tienen cardinalidad (1,1), se puede escoger cualquiera de los dos extremos para propagar la clave. En este caso, la elección puede depender de criterios como las frecuencias de acceso a las tablas.
- Si los dos extremos participan con una cardinalidad (0,1), crear una tabla que represente a la interrelación. El identificador de la tabla podrá ser el identificador de cualquiera de los dos extremos, y los atributos que sean clave primaria en uno de los dos extremos, serán clave foránea en la nueva tabla.

#### 7.1.5. Transformación de interrelaciones de dependencia y existencia.

El modelo relacional no distingue tipos de relaciones, por lo que las interrelaciones de dependencia y existencia se han de convertir en relaciones del mismo modo que las interrelaciones 1:M. Habitualmente, se propaga la clave de la tabla que representa a la entidad débil a la tabla que representa la entidad fuerte.

#### 7.1.6. Transformación de restricciones de entidades o atributos.

En el modelo E/R pueden estar expresadas restricciones de usuario. Estas restricciones se recogen en el esquema lógico estándar del siguiente modo:

- Si la restricción indica un rango de valores, se usa la cláusula **BETWEEN**.
- Si la restricción implica que un determinado atributo o conjunto de atributos solo puede tomar un valor de entre los pertenecientes a una lista, se emplea la cláusula **IN**.
- Si la restricción es de otro tipo, se puede utilizar la sentencia **CHECK** para comprobar el cumplimiento de la condición fijada, o la sentencia **CREATE ASSERTION** si la restricción afecta a más de una tabla.

### 7.1.7. Transformación de dependencias de identificación y existencia.

La transformación de estas dependencias se realizarán del mismo modo que el de las relaciones 1:M, es decir, no escribiendo ninguna relación que las represente y propagando la clave de la tabla que representa a la entidad fuerte a la tabla que representa a la entidad débil, en la cual será jugara el papel de clave foránea.

Para dicha clave foránea, no se admitirán los valores nulos, y se añadirá la condición de borrado y modificación en cascada (la eliminación o modificación de una ocurrencia de la entidad fuerte obligará a la eliminación o modificación de las ocurrencias de las entidades débiles que tiene asociadas).

Si la dependencia es en identificación, entonces la clave primaria de la tabla que representa a la entidad débil se formara mediante la concatenación de la clave primaria de la entidad débil y de la clave propagada que proviene de la entidad fuerte.

### 7.1.8. Transformación de restricciones en las interrelaciones.

Se utilizaran los mismos mecanismos que se han comentado para la transformación de restricciones de las entidades o de sus atributos (usando las condiciones CHECK o CREATE ASSERTION si la restricción afecta a una interrelación o a varias).

### 7.1.9. Transformación de generalizaciones (relaciones ISA).

Hay tres estrategias para llevar a cabo esta transformación:

- Transformar la entidad y sus subtipos en una sola tabla, la cual tendrá como atributos la concatenación de los atributos de la entidad y de los subtipos.
- Crear una tabla para la entidad generalizadora y una tabla por cada subtipo. Cada tabla tendrá como atributos los de su entidad o subtipo correspondiente. Esta es la opción que mejor mantiene la semántica del modelo E/R. En el ejemplo:

EMPLEADO\_PUBLICO (DNI, Nombre)

INTERINO (DNI, Inicio\_Contrato)

FUNCIONARIO (DNI, Toma\_Posesion)

- Crear una tabla para cada subtipo. Cada tabla tendrá como columnas los atributos del subtipo al que representa y los atributos comunes (los que posee la entidad generalizadora. En el ejemplo:

INTERINO (DNI, Nombre, Inicio\_Contrato)

FUNCIONARIO (DNI, Nombre, Toma\_Posesion)

### 7.1.10. Transformación de la dimensión temporal.

Se distinguen dos casos:

- Si la dimensión temporal aparece en el modelo E/R como una entidad, se transformará del mismo modo que el resto de las entidades.
- Si la dimensión temporal aparece en forma de atributos de una interrelación, estos atributos se ubicarán en la tabla que les corresponda al transformar la interrelación (bien en la tabla de la interrelación o bien en la tabla hacia la que se hayan propagado claves). Ahora bien, debe considerarse que estos atributos de tipo fecha pueden tener que formar parte de la clave primaria de la tabla en la que se ubiquen en función de la semántica de la situación que se representa.

### 7.1.11. Transformación de atributos derivados.

Los atributos derivados se transformaran en columnas de la entidad a la que pertenezcan (como el resto de los atributos). Además, se establecerá un disparador o un procedimiento almacenado que calcule el valor del atributo cada vez que se inserta una nueva fila en la tabla o cada vez que se modifique en una fila el valor de alguno de los atributos a partir de los cuales se calcula el atributo derivado.

### 7.1.12. Normalización del esquema obtenido.

Finalizada la transformación de el esquema conceptual en un esquema lógico, debe aplicarse a dicho esquema lógico un proceso de normalización, evitándose así las anomalías de inserción, modificación y borrado que provoca la redundancia. El proceso de normalización puede encontrarse comentado en el tema anterior de este mismo temario.

## 7.2. DISEÑO LÓGICO ESPECÍFICO.

Partiendo del esquema lógico obtenido en el apartado anterior, se elabora un esquema adaptado al sistema gestor de bases datos que se va a utilizar, creándose las tablas del esquema utilizando el Lenguaje de Definición de Datos propio de cada sistema. En las bases de datos relacionales, el Lenguaje de Definición de Datos habitual es el SQL, si bien existen pequeñas variaciones entre el SQL usado en cada sistema, que normalmente incluye pequeñas modificaciones o extensiones del lenguaje SQL-92 (que es el estándar ISO).

En el paso del modelo lógico estándar al modelo lógico específico de cada SGBD, puede encontrarse que el modelo lógico específico soporta todos los conceptos del modelo lógico estándar (del modelo relacional) o, por el contrario, que existen determinados aspectos del modelo lógico estándar que el modelo lógico específico no soporta. En este último caso, habrá que realizar un trabajo complementario de adaptación (bien añadiendo programación complementaria en el diccionario de datos del SGBD o bien haciendo que la puesta en practica de esas restricciones no soportadas por el modelo lógico del SGBD, la lleve a cabo el código de los programas que utilicen los datos de la BD).

Algunos de los aspectos del modelo lógico estándar que pueden tener que adaptarse son los siguientes:

- a) Dominios: el DDL del SGBD puede no incluir ninguna sentencia que nos permita crear dominios (los únicos dominios que reconoce automáticamente son los asociados a los tipos de datos predefinidos en el propio SGBD).

En este caso, habrá que elegir una de las dos opciones siguientes:

- Cuando se especifique la columna (dentro de la sentencia CREATE TABLE), escoger el tipo de datos predefinido que mejor se ajuste, fijar la longitud y añadir alguna restricción CHECK.
  - Crear una tabla de dominio, que contendrá una sola columna, y en la que cada fila será uno de los valores posibles del dominio. Una vez creada esta tabla, crear un procedimiento almacenado que compruebe que los valores que se intentan insertar en la columna son compatibles con el dominio que queremos establecer. La tabla de dominio será estática, es decir, sólo podrá ser modificada por el administrador de la base de datos. Lógicamente, la opción de construir una tabla de dominio sólo será válida si el dominio a construir es finito.
- b) Clave primaria: si el SGBD no incluye una cláusula PRIMARY KEY, debe conservarse la semántica dando los siguientes pasos:
1. Añadir la restricción NOT NULL en los atributos que formen parte de la clave primaria (debe recordarse que una clave primaria no admite valores nulos).
  2. Añadir la restricción UNIQUE al conjunto de atributos de la clave primaria (ya que una clave primaria no admite valores repetidos).
  3. Añadir a la tabla un índice construido sobre las columnas que forman parte de la clave primaria. Este índice se debe crear al crear la tabla y se debe destruir cuando la tabla sea eliminada.
  4. Documentar el esquema con un comentario que indique cuál es la clave primaria.
- c) Clave ajena: si el SGBD no incluye una cláusula FOREIGN KEY, debe conservarse la semántica dando los siguientes pasos:
1. Añadir la restricción NOT NULL en los atributos de la clave ajena que no admitan nulos (cuando la cardinalidad mínima de la interrelación original fuera de al menos uno).
  2. Hacer que los programas que trabajen con la Base de Datos implementen las restricciones de clave ajena (integridad referencial).
  3. Documentar el esquema con un comentario que indique que una columna o conjunto de columnas son clave ajena.

El resto de los aspectos del modelo lógico estándar no recogidos por el modelo lógico específico del SGBD suelen modelarse empleando procedimientos almacenados o triggers.

Finalizada la etapa del diseño lógico específico, habremos creado un esquema lógico específico en el SGBD. Esto quiere decir que ya tendremos una BD operativa en la que se podrán insertar, eliminar o modificar datos y sobre la cual podremos realizar consultas.



## 8. DISEÑO FÍSICO.

El diseño físico es la última etapa del proceso de creación de una BD. El objetivo de esta fase es obtener un esquema interno de la Base de Datos que cumpla lo mejor posible los objetivos de funcionamiento de la BD que los usuarios esperan. Más concretamente, se trata de:

- Disminuir el tiempo de respuesta de la BD (tanto el tiempo medio como la respuesta ante los picos de carga).
- Disminuir el espacio de almacenamiento utilizado.
- Incrementar la seguridad de la BD.

Estos objetivos del diseño físico no siempre son compatibles entre sí. Por ejemplo, para reducir el tiempo de respuesta de las consultas a una BD, puede ser necesario incrementar la redundancia de los datos (tener los mismos datos almacenados en varias tablas a la vez). Obviamente, esto incrementará el espacio de almacenamiento utilizado. Un buen diseño físico debe tener en cuenta para cada BD las necesidades de uso, establecer unos objetivos concretos, y, cuando estos objetivos sean contradictorios, priorizarlos y alcanzar un nivel de compromiso aceptable entre ellos.

Para llevar a cabo esta etapa, es preciso contar con información precisa sobre muchos aspectos de la BD que se va a crear y de la plataforma en la que se va a trabajar. El diseño físico comienza a realizarse cuando se ha recopilado información suficiente sobre:

- Los recursos software de los que se dispone.
- Los recursos hardware de los que se dispone.
- El esquema lógico específico de la BD.
- Políticas de seguridad de los datos.
- Estudio detallado de las aplicaciones que van a utilizar la Base de Datos y de las transacciones que van a generar.

El nivel físico de las BD no está estandarizado, por lo que la realización del diseño físico es dependiente del SGBD que se esté utilizando. Cada SGBD relacional definirá su propia estructura de archivos, índices, buffers de memoria, roles de seguridad y objetos de gestión del nivel físico, manipulándose este nivel a través de una extensión del lenguaje SQL estándar específica de cada sistema gestor.

De lo dicho en los párrafos anteriores se pueden extraer las siguientes conclusiones:

Al contrario que en el diseño lógico general, el enfoque del diseño físico no puede ser formal, sino casuístico, adaptado a cada SGBD y a cada BD utilizados. No hay recetas universalmente válidas, sino «buenas ideas» (heurísticas) sentadas en conceptos de almacenamiento, arquitectura de computadores, redes o algoritmia. Al no haber una estandarización del nivel físico, esas ideas deben ser puestas en práctica en cada caso concreto, probadas, evaluadas y, si es necesario, refinadas hasta alcanzar la situación final deseada. A este proceso de mejora del diseño físico se le conoce como ajuste de la BD o *tuning*.

Por último, es preciso comentar que no todos los SGBD tienen el mismo grado de flexibilidad en su nivel físico. En función del grado de manejo que permitan para el diseño físico, podemos distinguir tres tipos de SGBD:

1. Rígidos: el SGBD fija una estructura interna que apenas admite configuración. Esto asegura la independencia físico/lógica de la BD, pero es poco adaptable a cada situación concreta, lo que puede suponer una pérdida de eficiencia.
2. Flexibles: el SGBD permite que sea el Administrador de Bases de Datos el que diseñe toda la estructura interna. El diseño de toda la estructura interna es un trabajo extenso y complejo, y la toma de decisiones del administrador puede afectar a la independencia físico/lógica de los datos. Sin embargo, también es el enfoque más adaptable a cada necesidad concreta, con lo que es la alternativa con la que se podría obtener un mayor grado de eficiencia en el uso de la BD.
3. Semiflexibles: el SGBD proporciona una estructura inicial configurable a través de un conjunto de parámetros. La modificación de estos parámetros por parte del Administrador de Bases de Datos permite ir mejorando esa estructura interna, y por ende, el rendimiento de la Base de Datos. Esta opción ofrece un buen compromiso entre eficiencia e independencia físico/lógica, siendo habitual en los SGBD.

#### 8.1. METODOLOGÍA DE TRABAJO PARA LA OBTENCIÓN DEL DISEÑO FÍSICO.

Podemos dividir a la etapa del diseño físico en tres fases:

- Diseño de la representación física.
  - Análisis de las transacciones.
  - Selección del modo de almacenamiento en memoria secundaria.
  - Creación de índices secundarios.
  - Realización de Agrupamientos de tablas.
  - Realización de procesos de desnormalización.
  - Estimación de la necesidad de espacio en disco.
- Diseñar los mecanismos de seguridad.
  - Diseñar las vistas de los usuarios.
  - Diseñar las reglas de acceso.
- Monitorizar y ajustar del sistema.

### 8.1.1. Análisis de las transacciones.

Para realizar un buen diseño físico es necesario conocer las consultas y las transacciones que se van a ejecutar sobre la base de datos. Esto incluye tanto información cualitativa, como cuantitativa. Para cada transacción, hay que especificar:

- La frecuencia con que se va a ejecutar.
- Las relaciones y los atributos a los que accede la transacción, y el tipo de acceso: consulta, inserción, modificación o eliminación. Los atributos que se modifican no son buenos candidatos para construir estructuras de acceso.
- Los atributos que se utilizan en los predicados del WHERE de las sentencias SQL. Estos atributos pueden ser candidatos para construir estructuras de acceso dependiendo del tipo de predicado que se utilice.
- Si es una consulta, los atributos involucrados en el join de dos o más relaciones. Estos atributos pueden ser candidatos para construir estructuras de acceso.
- Las restricciones temporales impuestas sobre la transacción. Los atributos utilizados en los predicados de la transacción pueden ser candidatos para construir estructuras de acceso.

### 8.1.2. Selección de la organización del almacenamiento en memoria secundaria.

Las BD van a almacenar la información en dispositivos de almacenamiento secundarios (discos o cintas), los cuales se caracterizan por tener mayor capacidad que la memoria principal y por la no volatilidad de los datos. Sin embargo, son mucho más lentos que la memoria principal a la hora de recuperar información, por lo que es preciso realizar un estudio detallado sobre el modo de organizar la información en ellos, de modo que consigamos un rendimiento en tiempo de acceso ajustado a cada necesidad de uso de la BD.

Las alternativas de organización consisten básicamente en la elección del tipo de fichero o estructura de datos más adecuado para cada caso.

#### 8.1.2.1. Ficheros secuenciales.

Organizados de tal manera que cada registro es adyacente al siguiente registro. Esta relación de adyacencia puede ser física (direcciones físicas consecutivas) o lógica (haciendo que cada registro contenga un puntero al siguiente registro).

Los ficheros secuenciales no permiten el acceso directo a los datos, por lo que el acceso a los registros se realiza en el mismo orden en el que fueron introducidos en el fichero.

#### 8.1.2.2. Ficheros secuenciales indexados ISAM.

Es una estructura de fichero indexado en el que los registros se agrupan en bloques, y en el interior de dichos bloques están organizados secuencialmente.

El índice que se crea sobre el fichero contiene apuntadores a las direcciones de inicio de cada bloque, y el acceso a datos a través del índice se realiza de la siguiente manera:

1. Se localiza el índice de la clave que cumpla la condición de búsqueda.
2. Se accede al bloque apuntado por el nodo que contenía a la clave anterior.
3. Una vez dentro del bloque, el registro deseado se busca secuencialmente.

La organización de ficheros ISAM mantiene el equilibrio entre el tamaño de los índices y el tiempo de acceso de los registros.

Como el tamaño de los bloques está limitado, en estos ficheros hay una zona de desbordamiento en que se van a almacenar los registros que no se pueden guardar en el bloque que les corresponde cuando éste ya está lleno.

El uso de la zona de desbordamiento disminuye el rendimiento del sistema, puesto que se accede a ella tras buscar al registro en el bloque en el que le correspondería estar, y porque la búsqueda en la zona de desbordamiento es secuencial. Cuando el área de desbordamiento es muy grande, se reorganiza el fichero, realizándose una nueva división de bloques, ubicando a todos los registros en los bloques y reorganizando el índice de apuntadores a bloques.

#### 8.1.2.3. Árboles-B.

Estructura de indexación en forma de árbol equilibrado. El hecho de ser equilibrados (misma altura en todas sus ramas) permite minimizar el número de accesos a disco cuando se realiza una búsqueda: las búsquedas rápidas son una característica que distingue a los árboles-B.

En cuanto al almacenamiento, los árboles-B consiguen una gestión del espacio razonablemente buena, ya que si el árbol es de orden  $n$ , cada nodo debe tener al menos  $n/2$  claves (es decir, como mucho se desaprovecha la mitad del espacio de almacenamiento del índice, lo cual es fácilmente asumible con los recursos de almacenamiento de que se dispone hoy en día).

#### 8.1.2.4. Ficheros de acceso aleatorio empleando técnicas de hashing.

En estos ficheros se accede directamente a los registros mediante el valor de su clave (siendo la clave uno o más de los campos del registro). Para ello, se dispone de una función «hash» o de mapeado que permite calcular la dirección del registro a partir del valor de la clave. Este sistema es el que más rápido permite realizar una búsqueda de un registro concreto, pero sólo funciona para resolver consultas exactas (con todo el valor de la clave). Si la búsqueda es por rango o por patrón (por ejemplo, LIKE '%a'), no se puede aplicar la función hash.

#### 8.1.2.5. Criterios de elección entre las estructuras.

La elección de una estructura de organización dependerá del uso que se realice de los datos almacenados.

La siguiente tabla recoge las situaciones en las que se suele preferir cada estructura:

**TABLA 1. SITUACIONES APROPIADAS DE APLICACIÓN DE  
LOS CRITERIOS DE ORGANIZACIÓN DE LA MEMORIA SECUNDARIA**

ESTRUCTURA	SITUACIÓN
Ficheros secuenciales.	Archivos pequeños (es más costoso gestionar el índice que realizar las búsquedas secuenciales). Recuperaciones masivas de los datos (habrá que recorrer todos los datos). Acceso muy infrecuente a los datos y existencia de una sobrecarga de almacenamiento en el sistema (manteniendo el fichero secuencial se ahorra el espacio de almacenamiento del índice. Como el acceso a los datos es muy infrecuente, el rendimiento de la BD no se ve muy afectado).
HASH	Búsquedas exactas (con todo el valor de la clave).
Ficheros indexados (ISAM, Árboles-B o variantes)	En todos aquellos casos en los que no convenga utilizar ficheros secuenciales ni técnicas de HASH).

La elección entre un fichero ISAM y un árbol-B se realiza considerando los siguientes factores:

- Frecuencia de las actualizaciones de los datos: si la frecuencia de las actualizaciones es alta, debe elegirse un árbol-B, ya que los ficheros ISAM se irán degradando al irse añadiendo registros a la zona de desbordamiento.
- Elevado número de consultas concurrentes: si se realizan muchas consultas simultáneas sobre los datos indexados, el fichero ISAM debe ser la estructura elegida, ya que al ser su índice estático no se bloquea (facilita el acceso concurrente).
- Si se deben tener en cuenta los dos factores o no se conoce bien el entorno de explotación de la Base de Datos (existen dudas sobre el número de usuarios, la frecuencia de acceso a datos, etc.), la estructura elegida será el árbol-B, ya que es la más adaptable de las dos y sus aspectos desfavorables respecto a los ficheros ISAM tienen poca repercusión en el funcionamiento del sistema.

### 8.1.3. Creación de los índices secundarios.

Las BD relacionales indexan a cada tabla por su clave primaria, creándose automáticamente el índice de clave primaria en el mismo momento en el que se crea la tabla. Sin embargo, es frecuente

añadir índices adicionales a las tablas, para que hagan más rápidas las consultas que se realizan sobre determinados campos de esas tablas.

La introducción de un índice secundario en una tabla repercute negativamente en los tiempos de ejecución de la inserción o eliminación de registros, y supone un incremento del espacio de almacenamiento necesario para la tabla. El administrador de bases de datos debe ponderar en cada caso las pérdidas y ganancias de introducción de un nuevo índice, para así determinar si su creación es interesante. Algunas situaciones que hacen interesante la indexación son:

- Atributos de la relación a los que se accede con mucha frecuencia.
- Claves foráneas de la relación sobre las que es habitual realizar joins.

Algunas situaciones que desaconsejan la introducción de índices secundarios son:

- Tablas con pocas filas: el recorrido secuencial de las tablas sería muy breve, y la reducción del tiempo de acceso obtenida de la introducción del índice es muy escasa.
- Atributos cuyo valor se modifica muy a menudo: los índices utilizan como clave de búsqueda los valores de los atributos indexados. Si la modificación de esos valores es muy frecuente, habrá que reconstruir el índice cada poco tiempo, lo cual puede suponer un coste elevado si la tabla tiene muchas filas.
- Atributos con valores poco selectivos: los atributos en los que es muy habitual la repetición de su valor en distintas filas de la tabla no son buenos candidatos para la indexación. Después de recorrer el índice y localizar su clave de búsqueda, tendríamos todavía muchos registros con el mismo valor de clave, y habría que recorrer secuencialmente esos registros para encontrar el que se busca; en este caso, la introducción del índice no supondría ningún beneficio importante desde el punto de vista del tiempo de acceso. Por ejemplo, si tenemos una tabla de personas y deseamos buscar una persona en concreto, no tiene sentido indexar por un campo «Sexo», ya que aproximadamente la mitad de los registros de la tabla tendrían el valor «hombre» y la otra mitad el valor «mujer».

#### **8.1.4. Realización de agrupamientos de tablas (clustering).**

El clustering o agrupación de tablas es una técnica consistente en almacenar un grupo de tablas en una misma área de memoria secundaria. De este modo, los accesos simultáneos a las tablas agrupadas no obligan al SGBD a la búsqueda de los datos en zonas lejanas del almacenamiento secundario, lo que reduce el tiempo de resolución de esos accesos.

El clustering será una técnica a considerar si esos accesos simultáneos son frecuentes.

#### **8.1.5. Realización de procesos de desnormalización.**

El proceso de desnormalización consiste en introducir redundancias en un esquema lógico previamente normalizado (típicamente en 3FN o en FN de Boyce-Codd). Aunque ésta es una decisión de nivel lógico, se suele tomar por motivos de eficiencia de la BD (repetir datos en varias tablas evita navegar entre ellas para encontrarlos), es decir, la decisión viene motivada por cuestiones de implementación.

La desnormalización ralentiza las actualizaciones de datos, hace perder flexibilidad al esquema lógico (puede afectarnos si decidimos añadir nuevas tablas, por ejemplo) y complica la implementación de la BD y su documentación. Por tanto, es una alternativa que sólo se debe utilizar cuando el resto de las técnicas de diseño físico no nos proporcionan un rendimiento de las consultas de la BD que sea aceptable.

Entre las acciones de desnormalización que se pueden realizar, se pueden destacar:

- **Introducir atributos derivados:** los atributos derivados son aquellos cuyo valor se puede obtener realizando un conjunto de operaciones sobre atributos ya existentes en la BD. La introducción de un atributo derivado ayuda a obtener ese valor rápidamente en una consulta, puesto que no hay que realizar las acciones de cálculo del mismo, pero incrementa los costes de almacenamiento y de actualización de las tuplas de la BD (cada vez que se modifique el valor de un atributo que interviene en su cálculo, habrá que recalcular el valor del atributo derivado).
- **Fusionar tablas involucradas en relaciones uno a uno:** si el acceso conjunto a ambas tablas es muy habitual, puede ser interesante crear una nueva tabla cuyos atributos sean la concatenación de los atributos de las dos tablas, lo cual puede reducir su tiempo de acceso. Sin embargo, este cambio perjudicará a las operaciones join que se realicen entre cualquiera de esas dos tablas y una tercera, ya que las filas de la nueva tabla que se han de leer ocuparán más en memoria (la tabla fusionada tiene más campos), por lo que se recuperarán menos registros en cada lectura, lo cual obligará a un número de accesos a disco superior.
- **Introducir atributos duplicados en relaciones 1:N:** consiste en añadir atributos no clave de la tabla con cardinalidad 1 en la tabla con cardinalidad N. Así, cuando se realiza una operación de join entre las dos tablas que sólo involucra a los atributos duplicados, no será necesario recorrer la tabla con cardinalidad 1.

#### 8.1.6. Determinación del espacio de almacenamiento necesario.

Una vez obtenido el esquema de implementación definitivo, el administrador de la BD debe determinar el espacio de almacenamiento necesario para la BD. La cantidad de espacio que se determine será una estimación basada en el estudio de la cantidad de datos ya existente y que hay que cargar en la BD y en las estimaciones de crecimiento futuro de la BD.

La estimación del espacio puede verse afectada también por el nivel de seguridad que se desee para el almacenamiento de datos y por el grado de disponibilidad de la Base de Datos. Por ejemplo, si uno de los requisitos de la Base de Datos es que su disponibilidad sea 24x7 (24 horas al día, 7 días a la semana), es posible que se necesite usar una estructura de almacenamiento secundario en forma de RAID 1+0 o 0+1, lo que duplicaría el espacio de almacenamiento necesario para la BD.

### 8.2. DISEÑO DE LOS MECANISMOS DE SEGURIDAD DE LOS DATOS.

#### 8.2.1. Creación de las vistas de los usuarios.

Las vistas son un elemento de la BD que permiten a los usuarios ver los datos de una determinada forma (corresponden al nivel externo en la arquitectura ANSI/SPARC). El uso de las vistas permite:

- Reducir la complejidad el esquema lógico global, manteniendo un esquema único para todos los usuarios y adaptando ese esquema a las necesidades que tenga cada tipo de usuario.
- Mejorar el nivel de seguridad de la BD, ya que los usuarios no verán más datos de las tablas que aquellos que están incluidos en la vista.

### 8.2.2. Fijar las reglas de acceso de los usuarios.

En esta fase, se establecen los perfiles de usuario, donde cada perfil es el conjunto de acciones que cada tipo de usuario va a poder hacer sobre cada elemento o conjunto de elementos de la BD (tablas, filas, unidades lógicas de almacenamiento, etc.).

### 8.3. MONITORIZACIÓN Y AJUSTE DEL SISTEMA.

El diseño físico no se debe concebir como un proceso secuencial que finaliza cuando se ha encontrado una configuración valida. Las BD pueden sufrir variaciones a lo largo del tiempo, tanto en su tamaño, como en su esquema lógico global o en el uso que se desee hacer de ellas, lo que obligará a una monitorización continua de su rendimiento (para comprobar si se va degradando) y a la introducción de ajustes que permitan adaptarse a los cambios sufridos por la BD o solventar las pérdidas de eficiencia que se hayan producido.

Cada proceso de ajuste supone en cierta medida una reconstrucción del diseño físico que se haya realizado, por lo cual esta etapa de diseño volvería a comenzar de nuevo.

## 9. LA GESTIÓN DE LA CONCURRENCIA.

Veamos la gestión de la concurrencia en los SGBD. Para ello, empezaremos introduciendo la necesidad de dicha gestión y el concepto de transacción, pasando posteriormente a abordar los problemas que plantea la concurrencia (Lectura Fantasma, Lectura Sucia, etc.) y los mecanismos de resolución existentes.

### 9.1. NECESIDAD DE GESTIÓN DE LA CONCURRENCIA.

Los SGBD deben poder mantener la integridad de los datos que almacenan. Durante la vida de una BD, existen secuencias de acciones de escritura y lectura que pueden originar que la BD quede en un estado inconsistente si no se ejecutan todas las acciones de esa secuencia.

Para evitar este tipo de problemas, los SGBD utilizan las transacciones, que son unidades lógicas de proceso que se componen de una secuencia de acciones cuya ejecución es atómica, o se ejecutan todas o no se ejecuta ninguna. En un punto intermedio de una transacción, el estado de la Base de Datos puede ser inconsistente, siendo siempre consistente al principio y al final de la transacción.

El elemento del SGBD encargado de conseguir este objetivo es el Gestor de transacciones (transaction manager), el cual gestiona las peticiones de los usuarios en forma de transacciones.

Los Sistemas Gestores de Bases de Datos que existen en la actualidad ofrecen la posibilidad de ser usados en modo multiusuario, lo que implica que múltiples usuarios pueden trabajar a la vez con el



sistema como si fuera un recurso dedicado, sin apercibirse de la presencia de otros usuarios. Para lograr este efecto, los tiempos de respuesta del SGBD a todos los usuarios deben ser reducidos, lo que obliga a realizar una ejecución concurrente (simultánea) de las transacciones de cada usuario.

La ejecución atómica que caracteriza a las transacciones no garantiza que las pertenecientes a un programa de un usuario no puedan interferir en las transacciones que pertenecen a otros programas y que se están ejecutando al mismo tiempo. Conseguir que las transacciones no interfieran con el funcionamiento de otras transacciones se conoce como aislamiento de la transacción (isolation), y el conjunto de problemas que plantea conseguir ese aislamiento en un entorno multiusuario es conocido como el problema de la gestión de la concurrencia.

Además, durante el transcurso de una transacción, pueden producirse problemas en el SGBD que no permitan que la transacción concluya con éxito, lo que, en virtud del principio de «todo o nada» que acompaña a las transacciones, obliga al SGBD a estar preparado para deshacer las acciones que una transacción no concluida ha realizado sobre una base de datos. Es el problema de la recuperación y se gestiona añadiendo una nueva propiedad a las transacciones, que es la persistencia, y que consiste en que las modificaciones de datos que se realizan durante una transacción no se almacenan en la BD hasta que la transacción ha finalizado con éxito.

Resumiendo, una transacción ha de tener las siguientes propiedades:

- Atomicidad: las acciones de una transacción se ejecutan todas o ninguna.
- Consistencia: la base de datos se encuentra en un estado consistente antes de la ejecución de la transacción y debe estar en un estado consistente cuando la transacción termine.
- Aislamiento: la ejecución de una transacción no debe interferir en la ejecución de otras transacciones, la transacción debe ejecutarse como si estuviera aislada.
- Persistencia: los efectos de una transacción no son permanentes en la BD hasta que la transacción ha finalizado con éxito.

## 9.2. MANEJO DE TRANSACCIONES EN EL SGBD.

El gestor de transacciones es el componente funcional del SGBD que planifica y controla la ejecución de transacciones concurrentes en una BD.

Un gestor de transacciones se puede dividir conceptualmente en los siguientes elementos:

- Un contenedor de entrada: al que llegan las transacciones que se deben ejecutar.
- Un planificador (scheduler): que determina el orden en el que las transacciones de la cola van a ser ejecutadas. Cuando el planificador da paso a una transacción, ésta es enviada al gestor de datos, desde el cual se realizarán las operaciones de la transacción sobre la base de datos.
- Un contenedor de salida: al que llega la transacción cuando ha terminado de ejecutarse. Si la transacción ha finalizado correctamente, el gestor de transacciones realizará una operación commit, haciendo las modificaciones de la transacción persistentes en la base de datos.

Si la transacción no ha finalizado correctamente debido a algún problema, el gestor de transacciones ejecuta una operación abort y envía la transacción a un gestor de recuperación, el cual deshace las operaciones de la transacción y devuelve los datos al estado en que estaban antes de iniciarse la transacción. La transacción abortada será devuelta al contenedor de salida para volver a ser ejecutada

Con el fin de incrementar el rendimiento de las BD en un entorno multiusuario, las transacciones no se ejecutan secuencialmente una detrás de otra. El modelo de ejecución secuencial de las transacciones podría originar una situación en la que un programa corto se quedase bloqueado a la espera de la finalización de un programa largo.

En este caso, el usuario del programa debería sufrir un gran tiempo de espera para la realización de un conjunto reducido de operaciones, con lo que no percibiría a la BD como un recurso dedicado.

Para evitar estas situaciones, las transacciones se despachan concurrentemente; es decir, sus acciones se van a llevar a cabo de forma entrelazada. Sin embargo, el entrelazado de acciones de las transacciones puede originar conflictos, algunos de los cuales van a ser comentados en el punto siguiente.

La siguiente figura representa a un gestor de transacciones:

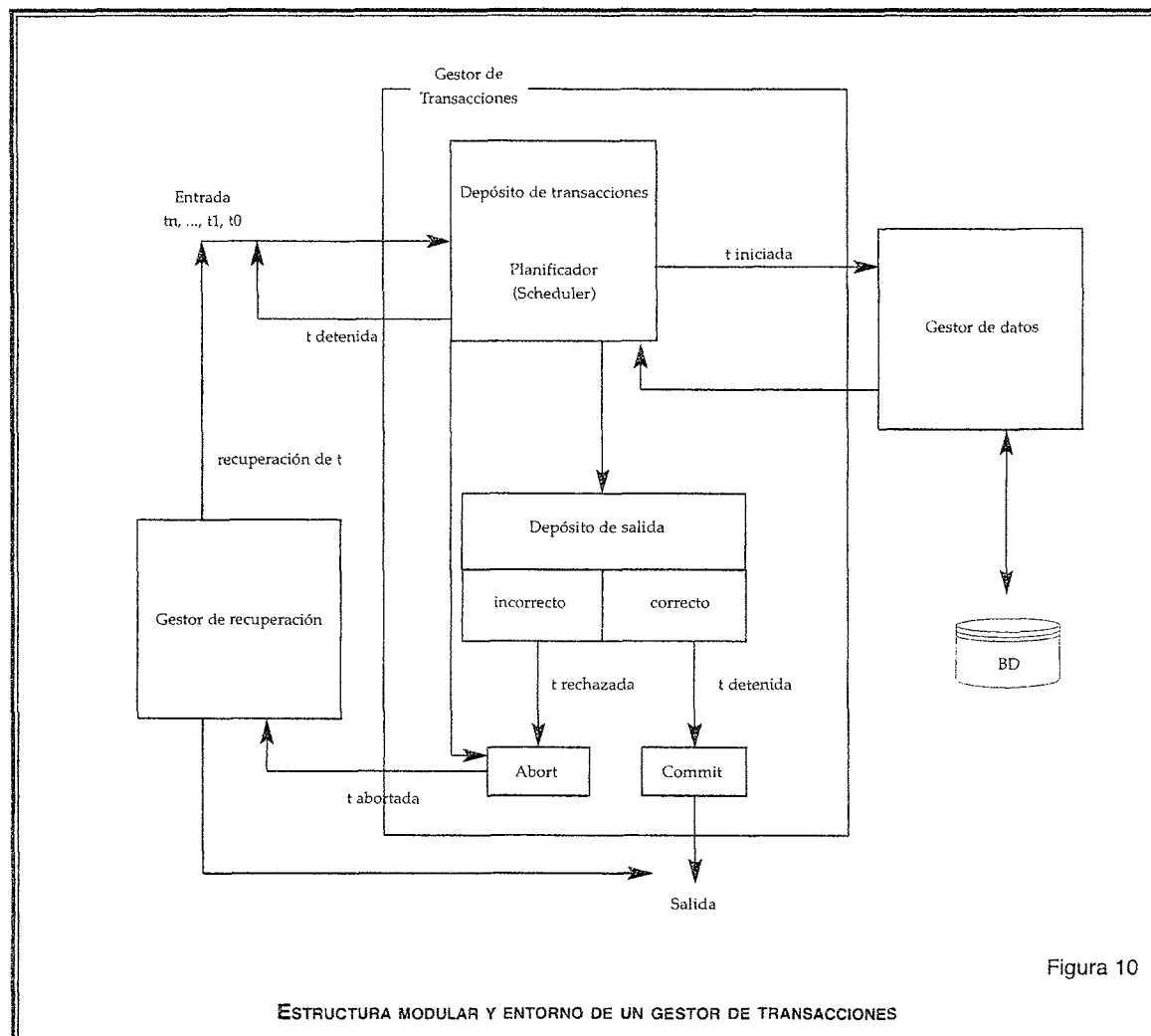


Figura 10

### 9.3. PROBLEMAS DE LA CONCURRENCIA.

#### 9.3.1. El problema de la actualización perdida.

El problema de concurrencia que ocurre cuando dos transacciones T1 y T2 trabajan en paralelo e intentan modificar el valor del mismo objeto de la BD. Ambas transacciones leen el valor del objeto antes de que la otra lo actualice.

En este caso, cada una de las transacciones modificará el valor del objeto en memoria y tratará posteriormente de escribirlo en la BD. El valor que se almacenará en la BD será el que tiene el objeto en la transacción que escriba más tarde, ya que el valor que ha escrito la primera transacción se sobrescribirá. Así, la actualización realizada por la transacción que ha escrito primero quedará sin efecto.

TRANSACCIÓN 1	TIEMPO	TRANSACCIÓN 2
Inicio	1	
r(x)	2	
	3	Inicio
	4	r(x)
u(x)	5	
	6	u(x)
w(x)	7	
Fin	8	
	9	w(x)
	10	Fin

NOTA: u(x) significa UPDATE, actualización de x.

#### 9.3.2. El problema de la lectura sucia.

Se produce cuando:

Una transacción T1 lee un valor de un objeto que ha sido modificado por otra transacción T2 y que todavía no se ha hecho persistente en la BD.

La transacción T2, que había modificado los datos no termina correctamente. En este caso, la transacción T1 está trabajando con un valor que es inconsistente.

Debe recordarse que los datos pueden encontrarse en un estado inconsistente en el transcurso de una transacción, por lo que la transacción primera puede haber leído dichos datos inconsistentes, utilizándolos para completar su secuencia de acciones.

TRANSACCIÓN 1	TIEMPO	TRANSACCIÓN 2
Inicio	1	
r(x)	2	
u(x)	3	
w(x)	4	
	5	Inicio
	6	r(x)
	7	u(x)
Abort	8	
	9	w(x)

### 9.3.3. El problema de la lectura fantasma o lectura no repetible.

Puede aparecer cuando:

Una transacción T1 está leyendo un conjunto de datos al tiempo que una transacción T2 los está modificando.

La transacción T1 lee algunos de los datos antes de que sean modificados por T2 y otros después de ser modificados por T2.

El resultado de la operación que realiza T1 se ve afectado por el hecho de combinar datos no actualizados por T2 con datos actualizados por T2.

TRANSACCIÓN 1	TIEMPO	TRANSACCIÓN 2
Inicio	1	
s:=0	2	
r(x)	3	
r(y)	4	
s:=s+x	5	
s:=s+y	6	
	7	Inicio
	8	r(x)
	9	z:=z-10
	10	w(z)
	11	r(x)
	12	x:=x+10
	13	w(x)
	14	commit
	15	Fin
r(z)	16	
s:=s+z	17	
Fin	18	

## 9.4. MECANISMOS DE RESOLUCIÓN DE CONFLICTOS.

Como ya hemos comentado en el punto anterior, la labor fundamental de un gestor de transacciones es manejar las transacciones que generan los programas de los usuarios de tal modo que se mantenga la consistencia de la BD. En un entorno multiusuario, el gestor de transacciones deberá conseguir mantener la consistencia cuando se ejecutan entrelazadamente las acciones que componen las transacciones de los distintos programas.

Este orden de ejecución lo establece el planificador (scheduler) del gestor de transacciones, llamándose plan de ejecución a cada posible secuencia de ejecución de un conjunto de acciones. Lógicamente, existen múltiples planes de ejecución posibles, siendo interesantes los planes que cumplen la condición de serializabilidad; un plan de ejecución A es serializable para un conjunto de transacciones T si existe un plan de ejecución secuencial A' que es equivalente a A.

O dicho de un modo menos formal, un plan es serializable si el resultado de la ejecución entrelazada de las acciones de esas transacciones es equivalente al resultado de ejecutar las transacciones en serie.

Sin embargo, la comprobación *a priori* de la serializabilidad de un plan es muy costosa (problema NP-completo), por lo que el gestor de transacción no trata de determinar si un plan de ejecución es serializable; en lugar de eso, el gestor de transacciones emplea diversos mecanismos de ejecución de las acciones que o eliminan los conflictos debidos a la concurrencia o permiten recuperar el estado consistente de la BD cuando se producen.

Podemos considerar los métodos de resolución de conflictos divididos en dos tipos:

- Métodos pesimistas: basados en la suposición de que los conflictos entre transacciones concurrentes ocurren con alta frecuencia, lo que obliga a realizar una serie de acciones para manejar esos conflictos.
- Métodos optimistas: basados en la suposición de que los conflictos entre transacciones se producen con escasa frecuencia.

### 9.4.1. Métodos pesimistas.

#### 9.4.1.1. Concepto de bloqueo y técnicas basadas en bloqueos.

Este mecanismo de resolución de conflictos se basa en el concepto de bloqueo, que consiste en que cuando una transacción T1 necesita realizar alguna acción sobre un objeto de la BD, debe solicitar al SGBD una reserva de ese objeto, no pudiendo ejecutar la acción hasta que la reserva no se ha producido. El SGBD sólo concederá esa reserva si otra transacción T2 no mantiene el mismo objeto reservado (en otras palabras, mientras una transacción tenga a un objeto de la BD reservado, el acceso del resto de las transacciones puede considerarse bloqueado).

En un momento dado, bien durante su transcurso o bien a su finalización, la transacción terminará las acciones que necesite hacer sobre el objeto de la BD, deshaciendo la reserva y quedando el objeto disponible para otras transacciones.

Las acciones de bloqueo o desbloqueo no tienen porque ejecutarse inmediatamente antes y después de la acción durante la cual queremos que el objeto este bloqueado. En el transcurso de una transacción se hacen múltiples lecturas y escrituras, pudiendo realizarse varias de ellas sobre el mismo objeto. Si se bloquease y desbloquease antes y después de cada acción, se incurrirá en un coste de gestión de los bloqueos muy alto, lo que repercutirá en el rendimiento de la BD (en sus tiempos de respuesta). Por tanto, una política de gestión de bloqueos debe llegar a un buen compromiso entre dos aspectos de signo contrario:

- Tiempo que el objeto (los datos) está bloqueado: cuanto mayor sea, más largo es el tiempo durante el cual otras transacciones no pueden realizar sus acciones sobre el conjunto de datos, lo que supone que han de estar a la espera del desbloqueo, retardándose su finalización.
- Tiempo empleado en gestionar los bloqueos: realizar muchas operaciones de bloqueo/desbloqueo repercute en el rendimiento de la BD, ya que hay que almacenar esos bloqueos, comprobar su compatibilidad, etc.

Así pues, la solución consiste en encontrar la manera de ejecutar un número elevado de acciones entre dos bloqueos, consiguiendo a la vez que la distancia entre bloqueos sea razonablemente pequeña.

Considerando el tipo de acción que se impide, podemos distinguir dos clases de bloqueo:

- Bloqueo de escritura (write-lock o wlock): impide que otras transacciones modifiquen un dato.
- Bloqueo de lectura (read-lock o rlock): impide que otras transacciones lean un dato.

La distinción entre bloqueos de escritura y bloqueos de lectura se establece porque interesa que el gestor de transacciones pueda hacer un manejo diferenciado de cada tipo:

Un bloqueo de lectura realizado por una transacción es compatible con un bloqueo de lectura de otra transacción. Dicho de otro modo, no existe ningún problema porque dos transacciones lean un objeto a la vez (y evitar esperas innecesarias supone incrementar el rendimiento de la BD).

Un bloqueo de escritura es incompatible con cualquier otro tipo de bloqueo, ya que no se puede permitir que una transacción lea un dato cuando otra transacción lo está modificando (como ya vimos en los problemas de la concurrencia) y, desde luego, y un objeto no puede ser sobrescrito a la vez.

Así pues, se puede establecer la siguiente tabla de compatibilidades entre bloqueos de lectura y escritura:

	read_lock	write_lock
read_lock	SI	NO
write_lock	NO	NO

Los SGBD también permiten manejar diferentes granularidades de los bloqueos: un bloqueo sobre distintas clases de objetos de la BD, como por ejemplo, una fila, un conjunto de filas o una tabla. De este modo, las transacciones pueden efectuar reservas del tamaño estrictamente necesario, no bloqueando innecesariamente partes de los objetos que pueden ser utilizadas simultáneamente por otras transacciones (ejemplo, una transacción puede necesitar hacer un bloqueo de escritura sobre las primeras  $n$  filas de una tabla, lo cual no debe impedir que otras transacciones realicen acciones de lectura o escritura sobre las  $n+1$  restantes).

Cuanto menor sea el tamaño de los objetos bloqueados (granularidad más fina), menor será el número de esperas que los bloqueos originarán. Sin embargo, una granularidad fina genera más situaciones de interbloqueo que una granularidad gruesa, por lo que será necesario hacer una selección cuidadosa del nivel de granularidad empleado.

#### A) El problema del interbloqueo.

El uso de técnicas basadas en bloqueo para la gestión de transacciones concurrentes puede conducir al problema denominado interbloqueo (o deadlock o abrazo mortal). El interbloqueo es una situación que se produce cuando dos transacciones se quedan esperando indefinidamente por un recurso que la otra transacción tiene bloqueada, de tal manera que ninguna de las dos accede nunca al recurso que la otra ha reservado.

Por tanto, el gestor de transacciones deberá implantar mecanismos que le permitan detectar esas situaciones de interbloqueo y resolverlas, existiendo tres tipos de estrategias de resolución:

- La predicción: consiste en la no generación de transacciones que puedan producir interbloqueo. Esto obligaría a analizar todas las transacciones a priori, lo cual produciría una gran sobrecarga en el sistema (todos los algoritmos de predicción son exponenciales, y por tanto no utilizables en un sistema con altas exigencias de rendimiento como es una base de datos).
- La prevención: basada en que las transacciones puedan renunciar a un bloqueo que hayan realizado previamente. Para ello, se detectan los casos en los que una transacción  $T_1$  tenga reservado un ítem y una transacción  $T_2$  intenta hacer un tipo de bloqueo sobre ese ítem que es incompatible con el bloqueo de  $T_1$ , aplicando técnicas de prevención, de entre las que se pueden destacar:

Wait-die: si  $T_2$  es más antigua, se le hace esperar. En caso contrario,  $T_2$  se cancela.

Kill-Wait: si  $T_2$  es más antigua, se cancela. En caso contrario, se hace esperar a  $T_2$ .

- La eliminación: es el modo más sencillo de los tres. Consiste en eliminar una de las transacciones bloqueadas, continuando la otra transacción su camino. El gestor de transacciones volvería entonces a generar la transacción eliminada y la reintroduciría en la cola de transacciones, siendo lo más probable que la otra transacción haya terminado o haya avanzado hasta un punto en el que no se repita la situación de interbloqueo.

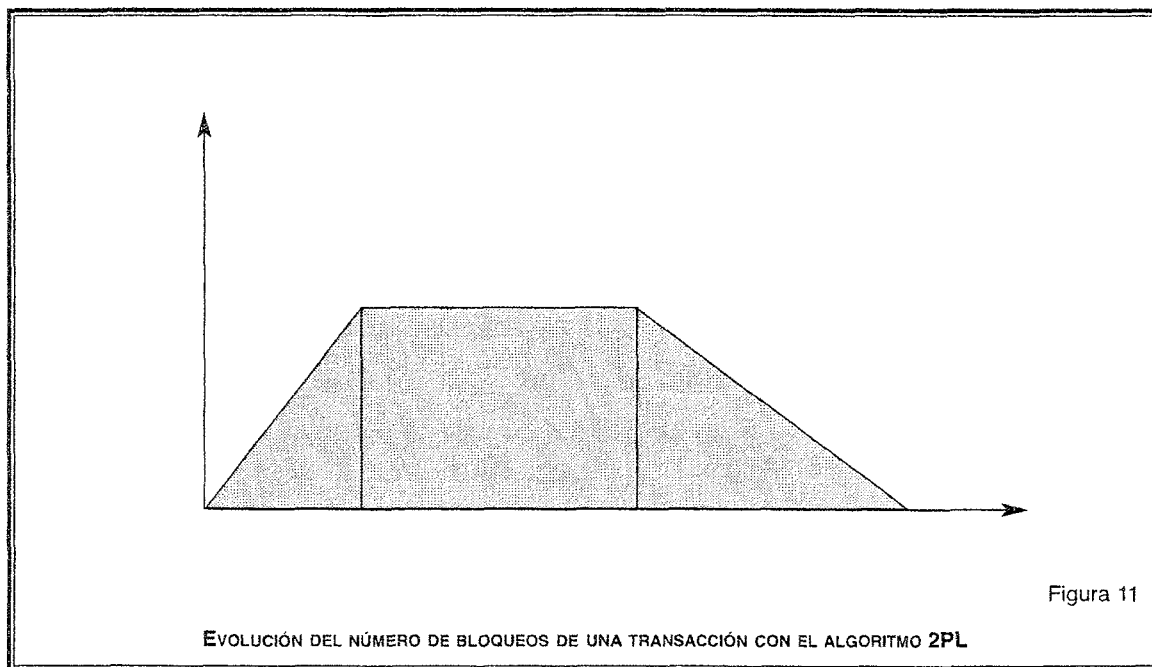
#### B) El protocolo de bloqueo en dos fases.

Es una implementación muy extendida de uno de los principios de la resolución de conflictos basada en bloqueos (técnica pesimista). El protocolo de bloqueo en dos fases define un conjunto de reglas so-

bre la aplicación de bloqueos que pueden ser utilizadas por el gestor de transacciones para gestionar las transacciones concurrentes.

Estas reglas son:

1. En una transacción sólo se va a escribir sobre un objeto una vez.
2. Si una transacción necesita leer y escribir, realizará en primer lugar el bloqueo de escritura. De esta manera, basta con realizar una única operación de desbloqueo en la transacción (un desbloqueo genérico que termina con el bloqueo de lectura y escritura).
3. Antes de realizarse las acciones de lectura o escritura, deben hacerse los bloqueos correspondientes, los cuales deben permanecer hasta que las acciones hayan finalizado.
4. Los bloqueos del mismo tipo sólo se realizan una vez por transacción.
5. En una transacción, una vez que se hace el primer desbloqueo, no se vuelve a hacer ningún bloqueo sobre ningún objeto. Esta norma caracteriza al bloqueo en dos fases, en la que se distingue una fase de crecimiento, donde van apareciendo nuevos bloqueos, y una fase de decrecimiento, que se inicia a partir del primer desbloqueo y dura hasta al final de la transacción, y en cuyo transcurso el número de bloqueos siempre decrece. Gráficamente, el comportamiento de este protocolo puede representarse así:



Entre las ventajas del protocolo de bloqueo en dos fases o 2PL, podemos citar dos:

- Es un protocolo seguro, es decir, nunca se van a producir las situaciones anómalas de concurrencia estudiadas en el punto anterior.
- Es un protocolo sencillo, fácil de implantar.



Entre sus desventajas, podemos citar:

- Los bloqueos tienden a ser grandes, lo que va a provocar incompatibilidades habituales con otras transacciones, derivándose de ello una pérdida de rendimiento.
- Este protocolo permite la aparición de interbloqueo.

#### 9.4.1.2. Time-stamping.

El time-stamping trata de conseguir la serializabilidad de los planes de ejecución haciendo que, cuando varias transacciones tengan que acceder a objetos comunes, lo hagan en distintos momentos. Este objetivo se consigue asignando una marca temporal distinta (time-stamp) a cada transacción.

Cuando una transacción intenta acceder a un objeto para leerlo o modificarlo, se comprueba previamente si ya ha sido accedido por otra transacción más joven. Existen tres modalidades de time-stamping:

##### A) Time-stamping básico.

1. Se almacena en cada objeto el time-stamp de la última transacción que lo ha leído (TSR) y el de la última que lo ha grabado (TSW).
2. Cuando una transacción T intenta leer un objeto, si su time-stamp es mayor o igual que el TSR del objeto, podrá leerlo, debiendo ser cancelada en caso contrario.
3. Si una transacción T intenta escribir un objeto, se pueden producir las siguientes situaciones:

$TS(T) \geq TSW(\text{Objeto})$ :

Si  $TS(T) \geq TSR(\text{Objeto})$ , podrá escribir. Si no, T tendrá que ser cancelada.

Si  $TS(T) < TSW(\text{Objeto})$  :

Si  $TS(T) \geq TSR(\text{Objeto})$ , T puede seguir adelante saltándose la grabación, ya que la versión que habría grabado no sería la última y no habrá sido leída por ninguna transacción (regla de escritura de Thomas). En caso contrario, T deberá ser cancelada.

El time-stamping básico no asegura que se eviten las anomalías de concurrencia, pero sí que impide los efectos de las anomalías de la Actualización Perdida y los la Lectura Sucia (ya que si la transacción no puede hacer un commit, tampoco podrán hacerlo las que hayan leído o escrito los objetos que ella ha actualizado).

##### B) Time-stamping dinámico.

Consiste en asignar un time-stamp a las transacciones cuando tratan de acceder a un objeto que ha sido leído o actualizado por otra transacción que no ha terminado todavía. De este modo, se evitan algunas de las cancelaciones de transacciones que se producen en el time-stamp básico. Su funcionamiento es el siguiente:

Cuando T1 intenta leer o actualizar un objeto actualizado por T2 o intenta actualizar un objeto que T2 ya ha leído:

1. Si T1 y T2 no tienen todavía time-stamp, se les asigna a cada una un time-stamp, cumpliéndose que  $TS(T1) > TS(T2)$ .
2. Si una de las dos no tiene time-stamp, se le asigna, respetándose de nuevo que  $TS(T1) > TS(T2)$ .
3. Si T1 y T2 tiene ya time-stamp y  $TS(T1) > TS(T2)$ , entonces continúa normalmente la ejecución de las dos transacciones. En caso contrario, se cancelará la transacción T1.

Al igual que el time-stamping básico, el time-stamping dinámico no asegura que se eviten las anomalías de concurrencia, pero sí que impide los efectos de las anomalías de la Actualización Perdi-da y la Lectura Sucia.

Además, es más eficiente que el time-stamping estático, ya que el número de cancelaciones que produce es menor.

#### C) Time-stamping multiversión.

Basada en almacenar versiones de los ítems de la BD, permite el acceso simultáneo a un ítem por parte de transacciones diferentes, de forma que los accesos de cada transacción a los ítems que necesita se hace siempre usando versiones consistentes de esos ítems.

#### 9.4.2. Métodos optimistas.

Basados en la suposición de que los conflictos entre transacciones se producen con escasa frecuencia. Por tanto, se acepta como válido cualquier plan de ejecución de transacciones, y es tras finalizar la ejecución del plan y antes de realizar el commit (los datos todavía no son persistentes en la BD) cuando se comprueba si se ha producido algún conflicto entre transacciones o se ha incumplido alguna condición de consistencia. Si estos problemas han aparecido, la transacciones afectadas se deshacen (abort) y vuelven a situarse en la entrada del gestor de transacciones para su ejecución.

En los métodos optimistas se puede considerar la ejecución de una transacción dividida en tres fases:

- La fase de lectura: en ella se produce la ejecución de las acciones de la transacción. Todos los objetos de la BD que se necesitan son leídos y escritos de un buffer de memoria local al programa, y por tanto las acciones no afectan al resto de las transacciones en ejecución.
- La fase de validación: se comprueba si las modificaciones introducidas por la transacción pueden ser hechas persistentes en la BD sin incumplir la condición de serializabilidad de la transacción.
- La fase de escritura: si la comprobación realizada durante la fase de validación es positiva, las acciones de la transacción se ejecutan sobre la BD. Si la comprobación es negativa, la transacción es abortada.

- **Técnica optimista básica.**

La técnica optimista básica determina el conjunto de transacciones que se han validado utilizando una técnica de time-stamping. Durante la fase de validación se analiza si hay algún ítem común y del mismo nivel de granularidad entre el conjunto de ítems leídos por una transacción T1 y el conjunto de ítems que han escrito las transacciones cuyo time-stamp se ha asignado entre el instante de inicio de y el instante de finalización de la fase de lectura de la transacción. Si no existe esa coincidencia, se añade un time-stamp a la transacción T1.

Esta técnica exige que haya como mucho una transacción en la fase de validación en cada instante de tiempo, por lo que la fase de validación actúa como cuello de botella del sistema de gestión de transacciones.

## 9.5. MÉTODOS DE CONTROL DE CONCURRENCIA BASADOS EN LA SEMÁNTICA.

Son métodos que, además de la sintaxis, utilizan la semántica de las acciones a realizar para determinar si un plan de ejecución es o no serializable. Es decir, no se limitan a comprobar si los accesos de las transacciones son de lectura o escritura, sino que analizan qué van a escribir concretamente las acciones de las transacciones y determinan si esas acciones pueden generar un conflicto o no.

## BIBLIOGRAFÍA

- Ingeniería del Software. Un enfoque práctico. Roger S. Pressman. Ed. McGraw-Hill.
- Concepción y Diseño de Bases de Datos. Adoración de Miguel y Mario Piattini. Ed. Ra-Ma.
- Análisis y Diseño detallado de Aplicaciones Informáticas de Gestión. Mario Piattini y otros. Ed. Ra-Ma.
- Ingeniería del Software. Ian Sommerville. Ed. Addison-Wesley.
- Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de Información. Métrica versión 3. Técnicas y Prácticas. Ministerio para las Administraciones Públicas.
- Metodología de Planificación y Desarrollo de Sistemas de Información. Métrica versión 2.1. Guía de Técnicas. Ministerio para las Administraciones Públicas. Ed. Tecnos.
- Temario de las pruebas selectivas para ingreso en el Cuerpo Superior de Sistemas y Tecnologías de la Información de la Administración del Estado. ASTIC.
- Temario de las pruebas selectivas para el acceso, por promoción interna, al Cuerpo de Gestión de Sistemas e Informática de la Administración del Estado. Ministerio para las Administraciones Públicas.
- Temario del Máster en Ingeniería del Software. Facultad de Informática. Universidad Politécnica de Madrid. Ed. Estudios Financieros.



