

BASES DE DATOS

I. Historia y Evolución de los SGBD

1. Introducción

Llama la atención el desarrollo sobresaliente de cualquier conjunto de datos organizados para su almacenamiento en la memoria de un ordenador o computadora en nuestros días, y logre un diseño para facilitar su mantenimiento y acceso de una forma estándar.

Hay dos buenas razones para la siguiente investigación. En primer lugar, el conocer los acontecimientos que dieron lugar al sistema gestor de base de datos nos proporciona cobertura detallada y comprensiva de su origen. En segundo lugar, si en algún momento fuera necesario convertir un sistema de gestión de base de datos, alcanzar cómo trabaja este sistema puede ser una ayuda esencial en el ámbito de los negocios, diseño e implementación de estrategias para la relación cliente/servidor.

El término bases de datos fue escuchado por primera vez en un simposio celebrado en California en 1963. En una primera aproximación, se puede decir que una base de datos es un conjunto de información relacionada que se encuentra agrupada o estructurada.

Desde el punto de vista informático, una base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos.

Por su parte, un sistema de Gestión de Bases de datos es un tipo de software muy específico dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan; o lo que es lo mismo, una agrupación de programas que sirven para definir, construir y manipular una base de datos, permitiendo así almacenar y posteriormente acceder a los datos de forma rápida y estructurada.

Actualmente, las bases de datos están teniendo un impacto decisivo sobre el creciente uso de las computadoras.

2. Orígenes

Los orígenes de las bases de datos se remontan a la Antigüedad donde ya existían bibliotecas y toda clase de registros. Además también se utilizaban para recoger información sobre las cosechas y censos. Sin embargo, su búsqueda era lenta y poco eficaz y no se contaba con la ayuda de máquinas que pudiesen reemplazar el trabajo manual.

Posteriormente, el uso de las bases de datos se desarrolló a partir de las necesidades de almacenar grandes cantidades de información o datos. Sobre todo, desde la aparición de las primeras computadoras, el concepto de bases de datos ha estado siempre ligado a la informática.

En 1884 Herman Hollerith creó la máquina automática de tarjetas perforadas, siendo nombrado así el primer ingeniero estadístico de la historia. En esta época, los censos se realizaban de forma manual.

Ante esta situación, Hollerith comenzó a trabajar en el diseño de una máquina tabuladora o censadora, basada en tarjetas perforadas.

Posteriormente, en la década de los cincuenta se da origen a las cintas magnéticas, para automatizar la información y hacer respaldos. Esto sirvió para suplir las necesidades de información de las nuevas industrias. Y a través de este mecanismo se empezaron

a automatizar información, con la desventaja de que solo se podía hacer de forma secuencial.

Las aplicaciones informáticas de los años sesenta acostumbraban a darse totalmente por lotes (batch) y estaban pensadas para una tarea muy específica relacionada con muy pocas entidades tipo.

Cada aplicación (una o varias cadenas de programas) utilizaba ficheros de movimientos para actualizar (creando una copia nueva) y/o para consultar uno o dos ficheros maestros o, excepcionalmente, más de dos.

Cada programa trataba como máximo un fichero maestro, que solía estar sobre cinta magnética y, en consecuencia, se trabajaba con acceso secuencial.

Cada vez que se le quería añadir una aplicación que requiriera el uso de algunos de los datos que ya existían y de otros nuevos, se diseñaba un fichero nuevo con todos los datos necesarios (algo que provocaba redundancia). Para evitar que los programas tuviesen que leer muchos ficheros.

A medida que se fueron introduciendo las líneas de comunicación, los terminales y los discos, se fueron escribiendo programas que permitían a varios usuarios consultar los mismos ficheros on-line y de forma simultánea. Más adelante fue surgiendo la necesidad de hacer las actualizaciones también on-line.

A medida que se integraban las aplicaciones, se tuvieron que interrelacionar sus ficheros y fue necesario eliminar la redundancia.

El nuevo conjunto de ficheros se debía diseñar de modo que estuviesen interrelacionados; al mismo tiempo, las informaciones redundantes (como por ejemplo, el nombre y la dirección de los clientes o el nombre y el precio de los productos), que figuraban en los ficheros de más de una de las aplicaciones, debían estar ahora en un solo lugar. El acceso on-line y la utilización eficiente de las interrelaciones exigían estructuras físicas que diesen un acceso rápido, como por ejemplo los índices, las multilistas, etc.

Estos conjuntos de ficheros interrelacionados, con estructuras complejas y compartidas por varios procesos de forma simultánea (unos on-line y otros por lotes), recibieron al principio el nombre de Data Banks, y después, a inicios de los años setenta, el de Data Bases. El software de gestión de ficheros era demasiado elemental para dar satisfacción a todas estas necesidades.

Por ejemplo, el tratamiento de las interrelaciones no estaba previsto, no era posible que varios usuarios actualizaran datos simultáneamente, etc.

La utilización de estos conjuntos de ficheros por parte de los programas de aplicación era excesivamente compleja, de modo que, especialmente durante la segunda mitad de los años setenta, fue saliendo al mercado software más sofisticado: los **Data Base Management Systems**, que aquí denominamos **Sistemas de Gestión de BD (SGBD)**.

En otras palabras, una base de datos es un conjunto estructurado de datos que representa entidades y sus interrelaciones. La representación será única e integrada, a pesar de que debe permitir utilizaciones varias y simultáneas.

Con todo lo que hemos dicho hasta ahora, podríamos definir el término BD. Una base de datos de un **SI (Sistema de Información)** es la representación integrada de los conjuntos de entidades instancia correspondientes a las diferentes entidades tipo del SI y de sus interrelaciones.

Esta representación informática (o conjunto estructurado de datos) debe poder ser utilizada de forma compartida por muchos usuarios de distintos tipos.

3. Década de los 60.

Posteriormente en la época de los sesenta, las computadoras bajaron los precios para que las compañías privadas las pudiesen adquirir; dando paso a que se popularizara el uso de los discos, cosa que fue un adelanto muy efectivo en la época, debido a que a partir de este soporte se podía consultar la información directamente, sin tener que saber la ubicación exacta de los datos.

En esta misma época se dio inicio a las primeras generaciones de bases de datos de red y las bases de datos jerárquicas, ya que era posible guardar estructuras de datos en listas y arboles.

Otro de los principales logros de los años sesenta fue la alianza de IBM y American Airlines para desarrollar **SABRE**, un sistema operativo que manejaba las reservas de vuelos, transacciones e informaciones sobre los pasajeros de la compañía American Airlines.

Y, posteriormente, en esta misma década, se llevo a cabo el desarrollo del IDS desarrollado por Charles Bachman (**que formaba parte de la CODASYL**) supuso la creación de un nuevo tipo de sistema de bases de datos conocido como modelo en red que permitió la creación de un estándar en los sistemas de bases de datos gracias a la creación de nuevos lenguajes de sistemas de información.

CODASYL (Conference on Data Systems Languages) era un consorcio de industrias informáticas que tenían como objetivo la regularización de un lenguaje de programación estándar que pudiera ser utilizado en multitud de ordenadores.

Los miembros de este consorcio pertenecían a industrias e instituciones gubernamentales relacionadas con el proceso de datos, cuya principal meta era promover un análisis, diseño e implementación de los sistemas de datos más efectivos; y aunque trabajaron en varios lenguajes de programación como COBOL, nunca llegaron a establecer un estándar fijo, proceso que se llevo a cabo por ANSI.

El Data Base / Data Comincations

IBM denominaba así (DB/DC) al software de comunicaciones y de gestión de transacciones de datos. Las aplicaciones típicas eran las reservas/compras de billetes a las compañías aéreas y de ferrocarriles. Más tarde, el mismo, se comenzará a utilizar en el mundo del mercado financiero y bursátil.

4. Década de los 70 – Sistemas Centralizados.

Los primeros SGBD de los años sesenta todavía no se les denominaba así. Estaban orientados a facilitar la utilización de grandes conjuntos de datos en los que las interrelaciones eran complejas.

El arquetipo de aplicación era el “**Bill of materials o parts explosión**”, típica en las industrias del automóvil o de la construcción, de naves espaciales y en campos similares, de aquellos momentos..

Estos sistemas trabajaban exclusivamente por lotes (batch).

Al aparecer los terminales de teclado, conectados al ordenador central (**Mainframes**) mediante una línea telefónica, se empiezan a construir grandes aplicaciones on-line transaccionales (**OLTP**).

Los SGBD estaban íntimamente ligados al software de comunicaciones y de gestión de transacciones.

Aunque para escribir los programas de aplicación se utilizaban lenguajes de alto nivel como Cobol o PL/I, se disponía también de instrucciones y de subrutinas especializadas para tratar las BD que

requerían que el programador conociese muchos detalles del diseño físico, y que hacían que la programación fuese muy compleja.

Puesto que los programas estaban relacionados con el nivel físico, se debían modificar continuamente cuando se hacían cambios en el diseño y la organización de la BD. La preocupación básica era maximizar el rendimiento: el tiempo de respuesta y las transacciones por segundo.

Por lo que respecta a la década de los setenta, Edgar Frank Codd, científico informático inglés conocido por sus aportaciones a la teoría de bases de datos relacionales, definió el modelo relacional a la par que publicó una serie de reglas para los sistemas de datos relacionales a través de su artículo “**Un modelo relacional de datos para grandes bancos de datos compartidos**”.

Este hecho dio paso al nacimiento de la segunda generación de los Sistemas Gestores de Bases de Datos.

Como consecuencia de esto, durante la década de 1970, Lawrence J. Ellison, más conocido como Larry Ellison, a partir del trabajo de Edgar F. Codd sobre los sistemas de bases de datos relacionales, desarrolló el Relational Software System, o lo que es lo mismo, lo que actualmente se conoce como Oracle Corporation, desarrollando así un sistema de gestión de bases de datos relacional con el mismo nombre que dicha compañía.

Inicialmente no se usó porque tuvo inconvenientes con el rendimiento, no podía competir con las bases de datos jerárquicas y de redes. Finalmente IBM desarrolló unas técnicas para construir un sistema de bases de datos relacionales eficientes, las cuales llamó System R; por otro lado Ingres se desarrolló en la UBC en los años de 1974 a 1977.

Ingres utilizaba un lenguaje de consulta, llamado QUEL, dando pie a la creación de sistemas como

Ingres Corporación, MS SQL Server, Sybase, PACE Wang, y Britton Lee-. Por su parte, el Sistema R utilizó el lenguaje de consulta Secuela, el cual ha contribuido al desarrollo de SQL / DS, DB2, Allbase, Oracle y SQL Non-Stop.

Posteriormente en la época de los ochenta también se desarrollará el **SQL (Structured Query Language)** o lo que es lo mismo un lenguaje de consultas o lenguaje declarativo de acceso a bases de datos relacionales que permite efectuar consultas con el fin de recuperar información de interés de una base de datos y hacer cambios sobre la base de datos de forma sencilla; además de analiza grandes cantidades de información y permitir especificar diversos tipos de operaciones frente a la misma información, a diferencia de las bases de datos de los años ochenta que se diseñaron para aplicaciones de procesamiento de transacciones.

Pero cabe destacar que ORACLE es considerado como uno de los sistemas de bases de datos más completos que existen en el mundo, y aunque su dominio en el mercado de servidores empresariales ha sido casi total hasta hace relativamente poco, actualmente sufre la competencia del SQL Server de la compañía Microsoft y de la oferta de otros Sistemas Administradores de Bases de Datos Relacionales con licencia libre como es el caso de PostgreSQL, MySQL o Firebird que aparecerían posteriormente en la década de 1990.

5. Década de los 80 – SGBD Relacionales.

Por su parte, a principios de los años ochenta comenzó el auge de la comercialización de los sistemas relacionales, y SQL comenzó a ser el estándar de la industria, ya que las bases de datos relacionales con su sistema de tablas (compuesta por filas y columnas) pudieron competir con las bases jerárquicas y de red, como consecuencia de que su nivel de programación era sencillo y su nivel de programación era relativamente bajo.

Los ordenadores minis, en primer lugar, y después los ordenadores micros, extendieron la informática a prácticamente todas las empresas e instituciones.

Los SGBD de los años sesenta y setenta (IMS de IBM, IDS de Bull, DMS de Univac, etc.) eran sistemas totalmente centralizados, como corresponde a los sistemas operativos de aquellos años, y al hardware para el que estaban hechos: un gran ordenador para toda la empresa y una red de terminales sin inteligencia ni memoria.

Esto exigía que el desarrollo de aplicaciones fuese más sencillo.

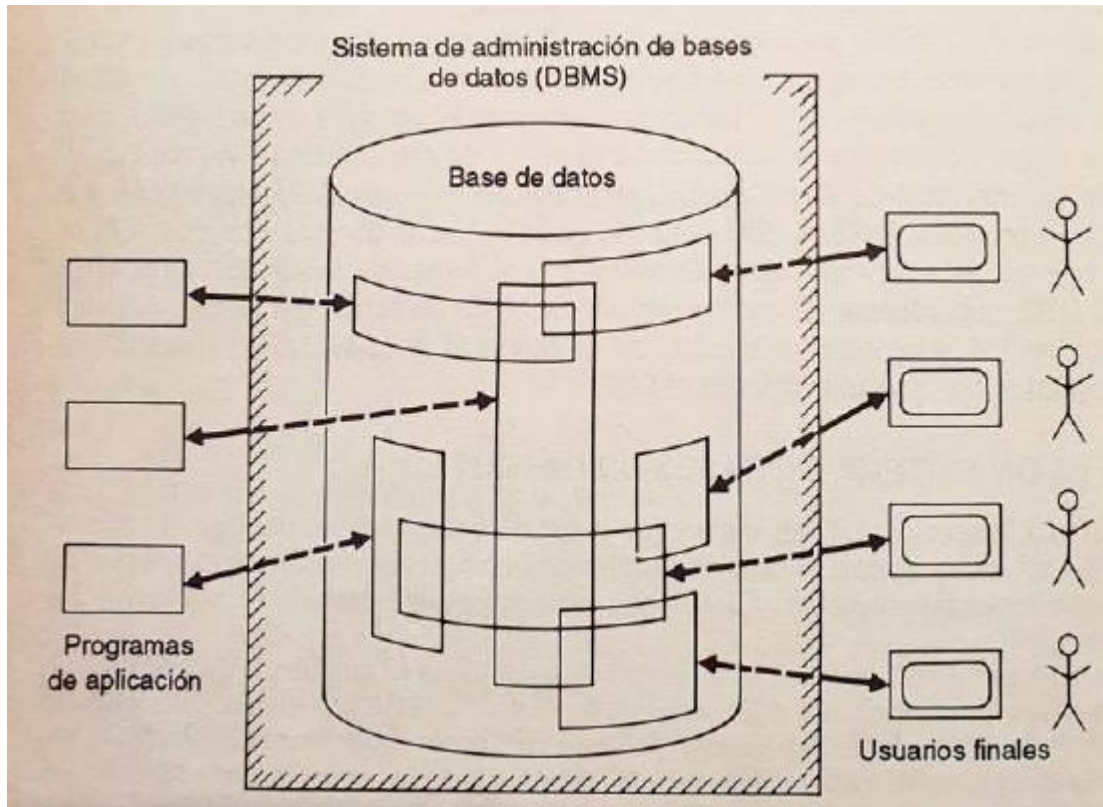
Los SGBD de los años setenta eran demasiado complejos e inflexibles, y sólo los podía utilizar un personal muy cualificado.

Los Ordenadores personales

Durante los años ochenta aparecen y se extienden muy rápidamente los ordenadores personales. También surge software para equipos monousuario (por ejemplo **dBase, Access**), con los cuales es muy fácil crear y utilizar conjuntos de datos, que se denominan **personal data bases**. Note que el hecho de dominar SGBD a estos primeros sistemas para PC es un poco forzado, ya que no aceptaban estructuras complejas ni interrelaciones, ni podían ser utilizados en una red que sirviese simultáneamente a muchos usuarios de diferentes tipos. Sin embargo, algunos, con el tiempo, se han ido convirtiendo en auténticos SGBD.

En la década de los años 80', se desarrolló el SQL (Structured Query Language), un lenguaje de consultas que permite consultar, valga la redundancia, con el fin de recuperar información de una base de datos y a su vez, hacer cambios sobre esa misma base, de forma sencilla. Permitía analizar gran cantidad de información y especificar varios tipos de operaciones con la misma información, a diferencia de los años anteriores, cuando se diseñaron aplicaciones de procesamiento de transacciones.

SQL comenzó a ser el modelo estándar de las industrias, con su base de datos bajo un sistema de tablas (filas y columnas), pudo competir con las bases jerárquicas y de redes, ya que su nivel de programación era sencillo y el nivel era relativamente bajo.



Estos sistemas de bases de datos relacionales fueron un éxito comercial, así como el aumento en la venta de ordenadores, estimulando el mercado de bases de datos, lo que provocó una caída importante en la popularidad de las redes y los modelos jerárquicos de bases de datos.

El ORACLE está considerado como uno de los sistemas de bases de datos más completos del mundo, su dominio en el mercado fue casi total hasta muchos años después, pero esto cambió con la aparición del SQL Server de Microsoft. La oferta de otros Sistemas Administradores de Bases de Datos Relacionales, como PostgreSQL, MySQL o Firebird aparecieron posteriormente en la década de 1990. Igualmente se da inicio a las bases de datos que se orientaban a los objetos.

6. Década de los 90: Distribución, C/S y 4GL

Al acabar la década de los ochenta, los SGBD relacionales ya se utilizaban prácticamente en todas las empresas. A pesar de todo, hasta la mitad de los noventa, cuando se ha necesitado un rendimiento elevado se han seguido utilizando los SGBD pre-relacionales.

A finales de los ochenta y principios de los noventa, las empresas se han encontrado con el hecho de que sus departamentos han ido comprando ordenadores departamentales y personales, y han ido haciendo aplicaciones con BD. El resultado ha sido que en el seno de la empresa hay numerosas BD y varios SGBD de diferentes tipos o proveedores.

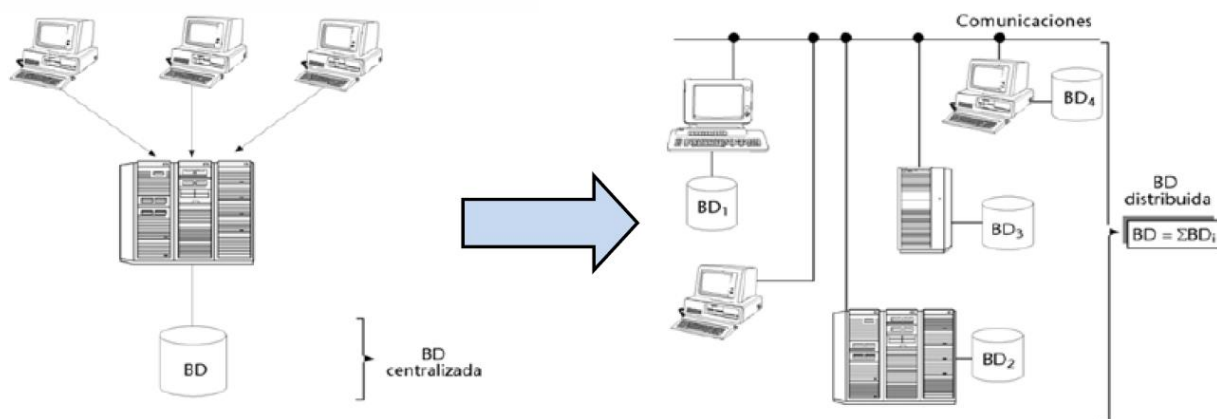
Este fenómeno de multiplicación de las BD y de los SGBD se ha visto incrementado por la fiebre de las fusiones de empresas.

La necesidad de tener una visión global de la empresa y de interrelacionar diferentes aplicaciones que utilizan BD diferentes, junto con la facilidad que las redes para la intercomunicación entre ordenadores, ha conducido a los SGBD actuales, que permiten que un programa pueda trabajar con diferentes BD como si se tratase de una sola. Es lo que se conoce como base de datos distribuida.

Esta distribución ideal se consigue cuando las diferentes BD son soportadas por una misma marca de SGBD, es decir, cuando hay homogeneidad.

Sin embargo, esto no es tan sencillo si los SGBD son heterogéneos. En la actualidad, gracias principalmente a la estandarización del lenguaje SQL, los SGBD de marcas diferentes pueden darse servicio unos a otros y colaborar para dar servicio a un programa de aplicación. No obstante, en general, en los casos de heterogeneidad no se llega a poder dar en el programa que los utiliza la apariencia de que se trata de una única BD.

Además de esta distribución “impuesta”, al querer tratar de forma integrada distintas BD preexistentes, también se puede hacer una distribución “deseada”, diseñando una BD distribuida físicamente, y con ciertas partes replicadas en diferentes sistemas.



Las razones básicas por las que interesa esta distribución son las siguientes:

Disponibilidad

La disponibilidad de un sistema con una BD distribuida puede ser más alta, porque si queda fuera de servicio uno de los sistemas, los demás seguirán funcionando. Si los datos residentes en el sistema no disponible están replicados en otro sistema, continuarán estando disponibles. En caso contrario sólo estarán disponibles los datos de los demás sistemas.

Coste

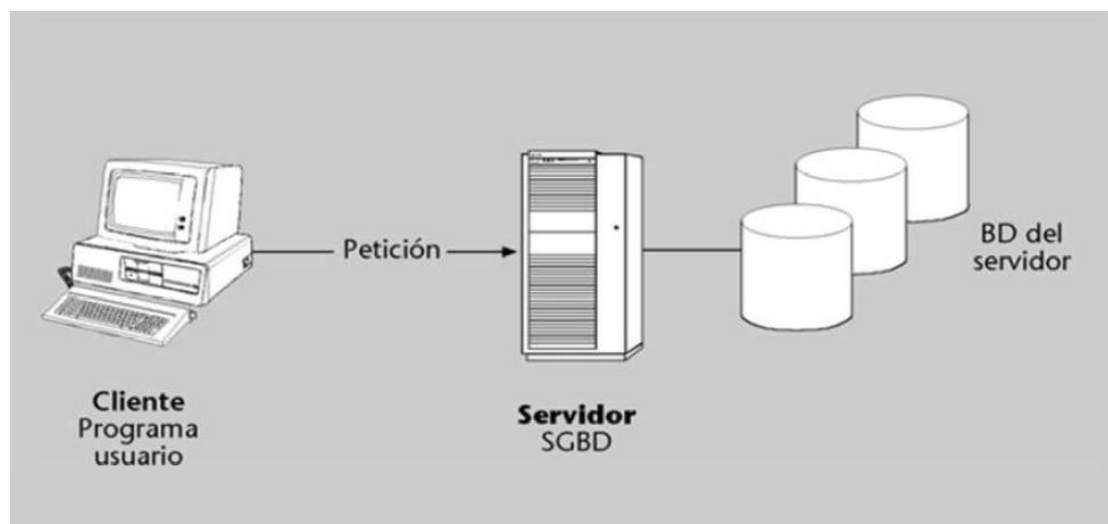
Una BD distribuida puede reducir el coste. En el caso de un sistema centralizado, todos los equipos usuarios, que pueden estar distribuidos por distintas y lejanas áreas geográficas, están conectados al sistema central por medio de líneas de comunicación. El coste total de las comunicaciones se puede reducir haciendo que un usuario tenga más cerca los datos que utiliza con mayor frecuencia; por ejemplo, en un ordenador de su propia oficina o, incluso, en su ordenador personal.

La tecnología que se utiliza habitualmente para distribuir datos es la que se conoce como entorno (o arquitectura) **cliente/servidor (C/S)**. Todos los SGBD relacionales del mercado han sido adaptados a este entorno.

La idea del C/S es sencilla. Dos procesos diferentes, que se ejecutan en un mismo sistema o en sistemas separados, actúan de forma que uno tiene el papel de cliente o peticionando un servicio y el otro el del servidor o proveedor del servicio.

Por ejemplo, un programa de aplicación que un usuario ejecuta en su PC (que está conectado a una red) pide ciertos datos de una BD que reside en un equipo UNIX donde, a su vez, se ejecuta el SGBD relacional que la gestiona. El programa de aplicación es el cliente y el SGBD es el servidor.

Un proceso cliente puede pedir servicios a varios servidores. Un servidor puede recibir peticiones de muchos clientes. En general, un proceso A que hace de cliente, pidiendo un servicio a otro proceso B puede hacer también de servidor de un servicio que le pida otro proceso C (o incluso el B, que en esta petición sería el cliente). Incluso el cliente y el servidor pueden residir en un mismo sistema.

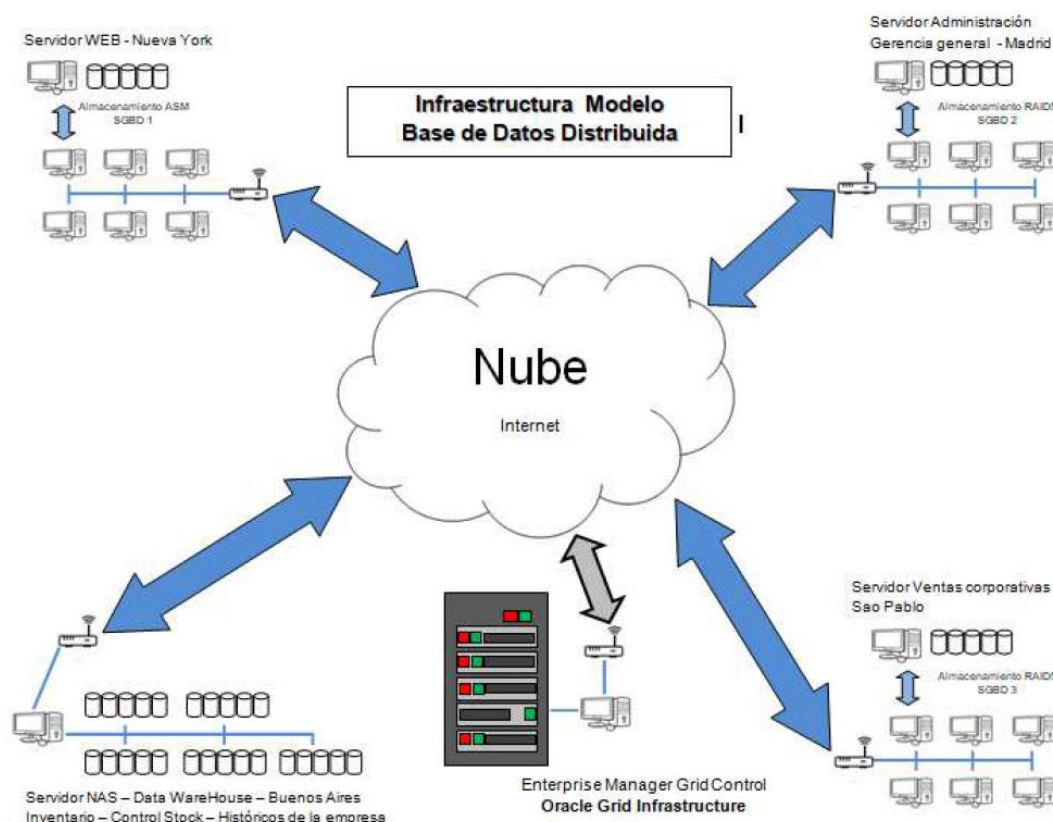


La facilidad para disponer de distribución de datos no es la única razón, ni siquiera la básica, del gran éxito de los entornos C/S en los años noventa.

Tal vez el motivo fundamental ha sido la flexibilidad para construir y hacer crecer la configuración informática global de la empresa, así como de hacer modificaciones en ella, mediante hardware y software muy estándar y barato.

El éxito de las BD, incluso en sistemas personales, ha llevado a la aparición de los Fourth Generation Languages (4GL), lenguajes muy fáciles y potentes, especializados en el desarrollo de aplicaciones fundamentadas en BD.

Proporcionan muchas facilidades en el momento de definir, generalmente de forma visual, diálogos para introducir, modificar y consultar datos en entornos C/S.



7. Tendencias actuales

Los tipos de datos que se pueden definir en los SGBD relacionales de los años ochenta y noventa son muy limitados. La incorporación de tecnologías multimedia –imagen y sonido – en los SI hace necesario que los SGBD relacionales acepten atributos de estos tipos.

Sin embargo, algunas aplicaciones no tienen suficiente con la incorporación de tipos especializados en multimedia. Necesitan tipos complejos que el desarrollador pueda definir a medida de la aplicación.

En definitiva, se necesitan tipos abstractos de datos: **TAD**. Los SGBD más recientes ya incorporaban esta posibilidad, y abren un amplio mercado de TAD predefinidos o librerías de clases. Esto nos lleva a la orientación a objetos (OO). El éxito de la OO al final de los años ochenta, en el desarrollo de software básico, en las aplicaciones de ingeniería industrial y en la construcción de interfaces gráficas con los usuarios, ha hecho que durante la década de los noventa se extendiese en prácticamente todos los campos de la informática. En los SI se inicia también la adopción, tímida de momento, de la OO.

La utilización de lenguajes como C++ o Java requiere que los SGBD relacionales se adapten a ellos con interfaces adecuadas.

La rápida adopción de la web a los SI hace que los SGBD incorporen recursos para ser servidores de páginas web, como por ejemplo la inclusión de SQL en guiones HTML, SQL incorporado en Java, etc.

Durante estos últimos años se ha empezado a extender un tipo de aplicación de las BD denominado Data Warehouse, o almacén de datos, que también produce algunos cambios en los SGBD relacionales del mercado.

A lo largo de los años que han trabajado con BD de distintas aplicaciones, las empresas han ido acumulando gran cantidad de datos de todo tipo. Si estos datos se analizan convenientemente pueden dar información muy valiosa. Por lo tanto, se trata de mantener una gran BD con información proveniente de toda clase de aplicaciones de la empresa (e, incluso, de fuera). Los datos de este gran almacén, el Data Warehouse, se obtienen por una replicación más o menos elaborada de las que hay en las BD que se utilizan en el trabajo cotidiano de la empresa. Estos almacenes de datos se utilizan exclusivamente para hacer consultas, de forma especial para que lleven a cabo estudios los analistas financieros, los analistas de mercado, etc.

Actualmente, los SGBD se adaptan a este tipo de aplicación, incorporando, por ejemplo, herramientas como las siguientes:

La creación y el mantenimiento de réplicas, con una cierta elaboración de los datos.

La consolidación de datos de orígenes diferentes.

La creación de estructuras físicas que soporten eficientemente el análisis multidimensional.

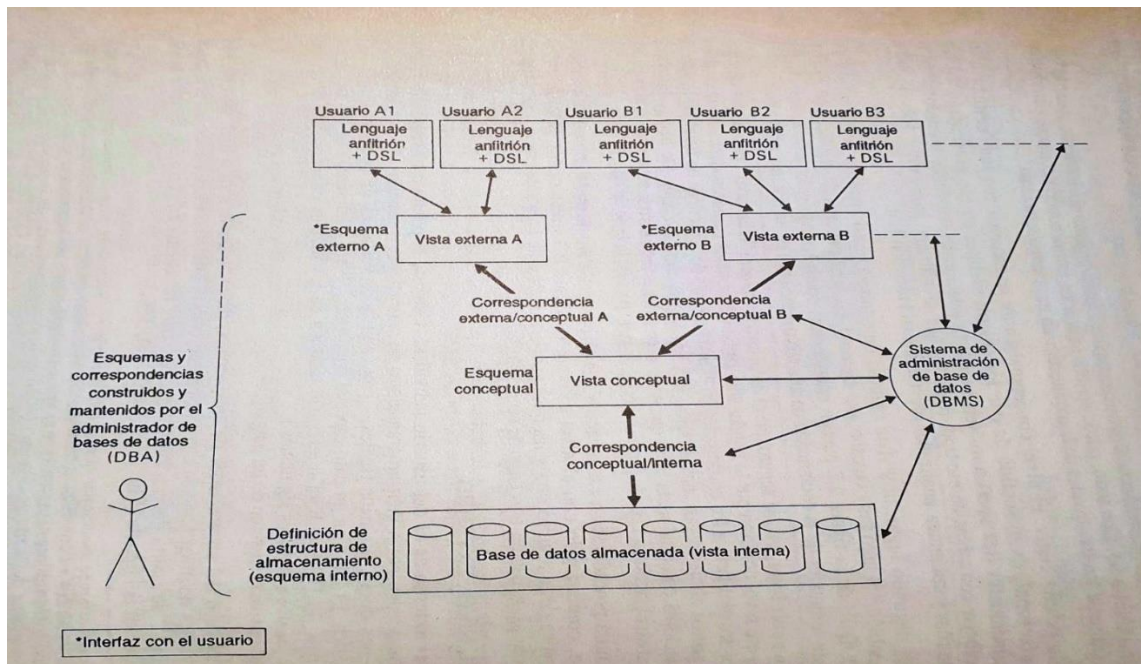
8. Gestores de Bases de Datos

Hoy día la mayoría de las bases de datos se presentan en formato digital, gracias a los avances tecnológicos en la informática y la electrónica. Esto ofrece un amplio abanico de soluciones al problema de almacenamiento de datos.

Los gestores de bases de datos, Database Management System o DBMS (SGBD) son programas que permiten almacenar y luego acceder a los datos de forma estructurada y rápida. Las aplicaciones más usadas son para gestiones de empresas e instituciones públicas, así como en entornos científicos, para almacenar la información experimental.

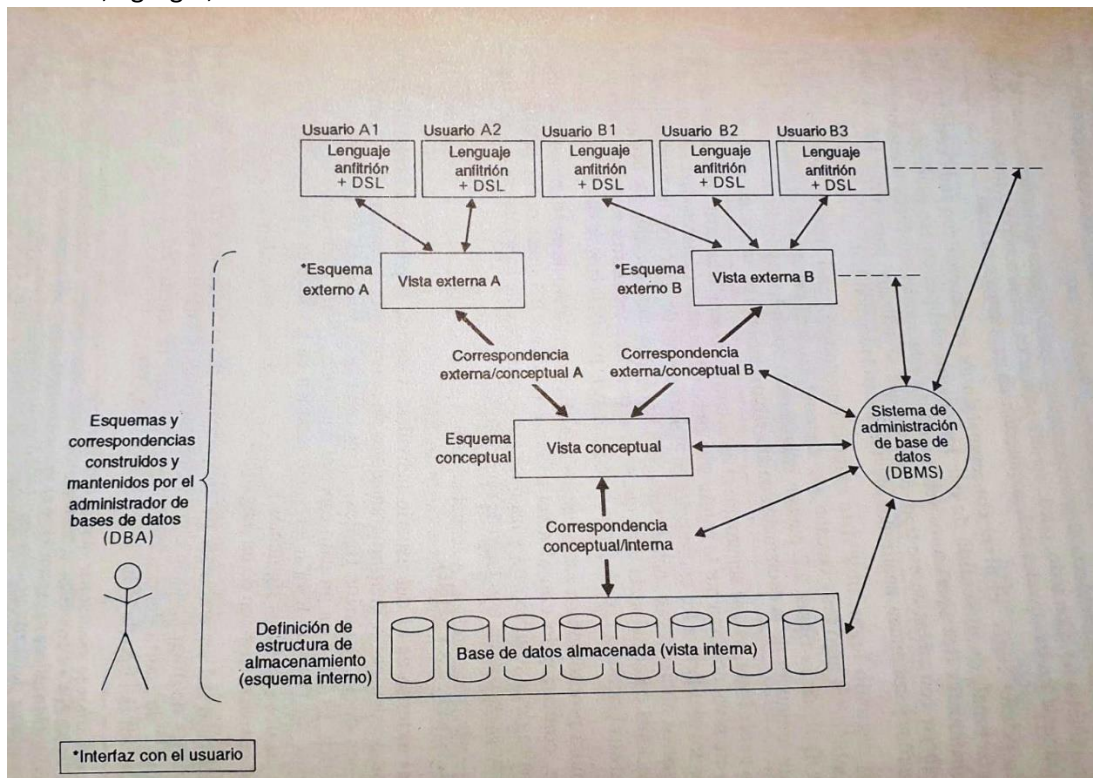
Una base de datos es un sistema compuesto por un conjunto de datos, los cuales están almacenados en discos, a los que se accede directamente y un conjunto de programas que regulen o manejen ese conjunto de datos.

Mientras que un sistema de Gestión de Bases de Datos es un software que sirve de interfaz entre la base de datos, el usuario y las aplicaciones que se utilizan.



Los mejores gestores de base de datos

El principal lenguaje de base de datos y el más utilizado desde que se conoce la programación de gestión, es el Structured Query Language (SQL). Este, de consulta estructurada, facilita el acceso a la gestión de las bases de datos relaciones, lo que permite realizar tareas en ellas y realizar consultas, que sirvan para obtener, agregar, eliminar o modificar información.



Para el desarrollo de este lenguaje hay que utilizar un gestor de base de datos, de los que hay muchos, unos de acceso libre y otros de pago. Veamos cuáles son, primeramente, los gestores de base de datos de pago:

Oracle

Es de los más confiables sistemas de gestión de base de datos relacional, además del más usado. Es propiedad de Oracle Corporation y fue desarrollado en 1977. Se accede directamente a los objetos, a través del lenguaje de consulta SQL, es muy utilizado en las empresas, con un componente de red que permite la comunicación a través de las redes.

Su versatilidad le facilita ejecutarse en casi todas las plataformas existentes, Windows, Unix, Linux, MAC OS, entre otros.

SQL Server

En competencia directa a Oracle, está SQL Server de Microsoft. Los dos ocupan gran parte del mercado en el sector de base de datos. Son muy parecidos en algunas de sus características y funciones, aunque tienen sus marcadas diferencias.

SQL Server se ejecuta en Transact-SQL, esto es un grupo de programas que pueden añadir características al programa, como tratamiento de errores y excepciones, extracción de datos de la web en forma directa, procesamiento de datos, uso de distintos lenguajes de programación y

otros más, que lo hacen un gestor muy completo y competitivo. Su carácter administrativo es otro valor agregado, tanto en sus funciones y seguridad, como en su flexibilidad.

Gestores de base de datos de acceso libre

Dos de los principales y más utilizados gestores de pago, que son de acceso libre (Open Source) son los siguientes:

MySQL

Este es de simple instalación y actúa de lado del cliente o servidor, es de código abierto y tiene licencia comercial disponible. Pertenece a Oracle Corporation y gestiona las bases de datos relacionales, con funciones multiusuario y es el más usado dentro del software libre. Requiere de poca memoria y procesador para su funcionamiento, lo que se traduce en mayor velocidad en sus operaciones. Se usa principalmente para el desarrollo Web.

Fire Bird

De gran potencia y muy sencillo a la vez, este sistema de gestión de base de datos relacional SQL, es uno de los mejores gestores Open Source (Código abierto) o libres. Es compatible con Windows y Linux.

Tiene buen soporte para los procedimientos almacenados, las transacciones compatibles con ACID y con los métodos de acceso múltiple como Nativo, Python, .NET, etc...

Como vemos, son múltiples las posibilidades que tenemos de acceso a gestores de base de datos, tanto adquiriendo licencias de pago como acudiendo a software libre. En función de los gustos, formas de trabajar y necesidades de cada uno, seguro encontraremos distintos gestores de base de datos que pueden satisfacernos en pro de nuestro trabajo.

II Introducción a las Bases de Datos

1. Etapas de diseño

La metodología de diseño de bases de datos relacionales se ha consolidado a lo largo de los años satisfaciendo las propiedades de generalidad (independencia de la plataforma hardware/software), calidad del producto (precisión, completitud y eficacia) y facilidad de uso.

Consta de las siguientes etapas:

- **1. Diseño conceptual.**

Su objetivo es definir las entidades y las relaciones entre ellos de forma abstracta, sin centrarse en ningún modelo lógico en concreto (como el relacional, el orientado a objetos, el jerárquico o el de red).

Herramienta: Modelo conceptual de datos. Se usa alguna variante del modelo entidad-relación para las bases de datos relacionales.

Resultado: Esquema conceptual de la base de datos.

- **2. Diseño lógico.**

Su objetivo es definir el esquema de la base de datos según el modelo que implementa el SGBD objetivo.

Herramienta: Modelo lógico de datos. Se usa el modelo lógico que implemente el sistema de gestión de bases de datos objetivo, pero es independiente de los aspectos físicos. Se usan técnicas formales para verificar la calidad del esquema lógico; la más usual es la normalización. En el modelo relacional se usan las tablas.

Resultado: Esquema lógico de la base de datos.

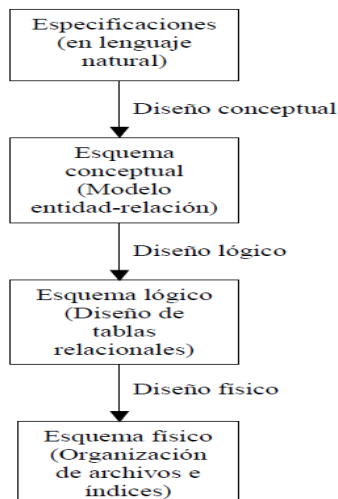
- **3. Diseño físico.**

Su objetivo es definir el esquema físico de la base de datos de forma que se den todas las instrucciones para que un DBA pueda implementar la base de datos sin ninguna ambigüedad. Se considera el rendimiento como un aspecto que no se ha tratado en las etapas anteriores.

Herramienta: Modelo físico de datos. Se consideran todos los detalles de la implementación física: organización de archivos e índices para el SGBD considerado.

Resultado: Esquema físico de la base de datos.

La siguiente figura muestra resumido el ciclo de desarrollo clásico de bases de datos:



2. Diseño conceptual

En este apartado se estudia el modelo entidad-relación que permite diseñar el esquema conceptual de una BD, y es muy adecuado para las BDs relacionales. Su resultado es un diagrama entidad-relación.

A lo largo de este apartado se usará un ejemplo de aplicación correspondiente a las necesidades de una secretaría de un centro docente, en la que hay alumnos matriculados en asignaturas y profesores que las imparten en ciertas aulas. Los alumnos consiguen una nota determinada en cada asignatura en que están matriculados.

Conceptos

- **Entidad:** Es el menor objeto con significado en una instancia. Constituye una entidad. Igual sucede con cada asignatura concreta, cada profesor, etc. En el caso del enfoque "clásico" correspondería a cada registro guardado en un fichero.

Por ejemplo, para el diseño de una BD de la secretaría de un centro docente, el alumno con los siguientes datos:

DNI = 01234567Z,
Nombre y apellidos = Manuel Vázquez Prieto,
Teléfono = 91-12345678
Domicilio = Calle del Jazmín 7, 4 Izq.

- **Atributo:** Es cada uno de los componentes que determinan una entidad. Cada atributo tiene asociado un dominio: el conjunto de valores que puede tomar. La entidad del ejemplo anterior viene determinada por los valores de sus atributos DNI, Nombre y Apellidos, Teléfono, Domicilio y COU.

En el enfoque clásico serían los campos de los registros.

- **Atributos monovalorados y multivalorados:** Los atributos multivalorados son los que pueden contener más de un valor simultáneamente, y monovalorados a los que sólo pueden contener un valor.

Por ejemplo, una persona puede tener varios números de teléfono (casa, trabajo, móvil) y puede que nos interese tenerlos todos. En este caso haremos de teléfono un atributo multivalorado.

- **Atributos simples y compuestos:** Un atributo es compuesto cuando puede descomponerse en otros componentes o atributos más pequeños, y simple en otro caso

Por ejemplo, en el caso del domicilio puede que nos interese descomponerlo a su vez en calle, el número y la ciudad por separado.

- **Clave:** Es un atributo o conjunto de atributos cuyos valores identifican unívocamente cada entidad. Por ejemplo, DNI es un atributo clave del tipo de entidad Alumnos. Esto significa que los valores de la clave no se pueden repetir en el conjunto de entidades. En el ejemplo anterior ningún DNI se debería repetir en una instancia del tipo de entidad Alumnos.

El concepto de clave distingue tres claves diferentes:

Superclave. Es cualquier conjunto de atributos que pueden identificar unívocamente a una tupla.

Clave candidata. Es el menor conjunto de atributos que puede formar clave. Puede haber varias en una tabla.

Clave Primaria. Es la clave candidata que distingue el usuario para identificar unívocamente cada tupla. Es importante en cuanto al aspecto del rendimiento, como se verá en el apartado dedicado al diseño físico.

- **Tipo de entidad.** Es el conjunto de entidades que comparten los mismos atributos (aunque con diferentes valores para ellos).

Por ejemplo, Alumnos será un tipo de entidad que representa cualquier conjunto de entidades en el que todas tengan como atributos

DNI, Nombre y Apellidos, ... y valores dentro de los dominios correspondientes. Asignaturas será otro tipo de entidad, etc.

Intuición: En el enfoque "clásico" sería el tipo de los registros.

Estamos describiendo el esquema de la base de datos.

- **Relación.** Es una correspondencia entre dos o más entidades. Se habla de relaciones binarias cuando la correspondencia es entre dos entidades, ternarias cuando es entre tres, y así sucesivamente.

Por ejemplo, la relación (José García, Bases de datos) es una relación entre dos entidades que indica que el alumno José García está matriculado en la asignatura Bases de datos.

- **Tipos de relación.** Representan a todas las posibles relaciones entre entidades del mismo tipo.

Por ejemplo, el tipo de relación matrícula relaciona el tipo de entidad alumnos con el tipo de entidad asignaturas.

Observaciones:

- Las relaciones también pueden tener atributos. Por ejemplo, Matrícula puede tener el atributo Nota que indica la nota que el alumno ha obtenido en una asignatura determinada.

Es posible que el mismo tipo de entidad aparezca dos o más veces en un tipo de relación. En este caso se asigna un nombre a cada papel que hace el tipo de entidad en el tipo de relación. Por ejemplo, algunos profesores tienen un supervisor, por lo que se define un tipo de relación Supervisa que relaciona profesores con profesores, el primero tendrá el papel de supervisor y el segundo de supervisado.

Diagramas entidad-relación (E-R)

El diseño del modelo E-R a partir del análisis inicial no es directo. A un mismo análisis le corresponden muchos diseños "candidatos". Hay varios criterios, pero ninguno es definitivo. De un buen diseño depende:

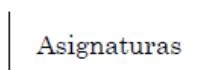
Eficiencia: Es muy importante en las BD cuando se manejan grandes cantidades de datos.

Simplicidad del código: Se cometen menos errores.

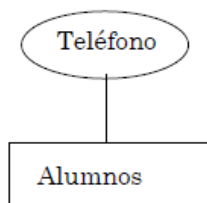
Flexibilidad: Se refiere a que el diagrama sea fácil de modificar.

Los componentes básicos de los diagramas E-R son los atributos, los tipos de entidades y los tipos de relaciones.

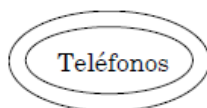
- Tipos de entidades: Rectángulos.



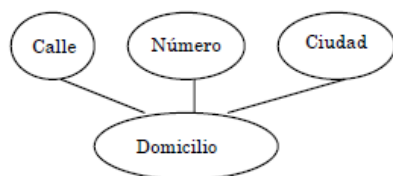
- Atributos: Elipses. Se conectan mediante líneas a los tipos de entidades o tipos de relación.



- Atributos multivalorados: Una elipse con doble línea:



- Atributos compuestos. Los componentes de un atributo se representan a su vez como atributos:



- Tipos de Relación: Rombos conectados a los tipos de entidades que relacionan.



Elección de los tipos de entidad y sus atributos

De la especificación del problema de la secretaría se deduce que va a haber un tipo de entidad alumnos, pero no cuáles son sus atributos. ¿Debe incluir las asignaturas en las que está matriculado? La respuesta es no y hacerlo así sería un error grave. Aparte de la idea 'filosófica' (cada asignatura es un objeto con significado propio, es decir, una entidad), al mezclar en una sola entidad alumnos y asignaturas cometemos cuatro errores:

1. Un alumno no tiene una asignatura asociada sino un conjunto de asignaturas asociadas. En cambio, sí tiene un DNI asociado, una dirección asociada, etc. Por tanto las entidades serán de la forma

**{DNI=12345678V, Nomb.Ape=Luis Martínez, Telf.=01234567,
Cod=MD, Título=Matemática Discreta, Créditos=9}**
**{DNI=12345678V, Nomb.Ape=Luis Martínez, Telf.=01234567,
Cod=IS, Título=Ingeniería del Software, Créditos=X}**
**{DNI=12345678V, Nomb.Ape=Luis Martínez, Telf.=01234567,
Cod=LPI, Título=Laboratorio de programación I, Créditos=X}**

Hay redundancia en la información de alumnos: se repite en cada entidad.

2. Las asignaturas son siempre las mismas, con lo que por cada alumno que se matricula en la misma asignatura hay que repetir toda la información:

**{ DNI=12345678V, Nomb.Ape=Luis Martínez, Telf.=01234567, ... ,
Asignaturas={ {Cod=MD, Título=...}, {COD=IS,Título=...},
{Cod=LPI,Título=...} } }**
**{ DNI=0000001, Nomb.Ape=Eva Manzano, Telf.=01234567, ... ,
Asignaturas={ {Cod=MD, Título=...}, {COD=IS,Título=...},
{Cod=BDSI,Título=...} } }**

En este caso hay redundancia en la información de las asignaturas.

3. Por cada profesor hay que apuntar las asignaturas que imparte. La información de las asignaturas debe estar por tanto relacionada con la de los profesores, pero ya está incluida con los alumnos. Hay que repetir la información de las asignaturas por lo que se consigue más redundancia.

4. No se pueden guardar los datos de una asignatura hasta que no se matricule un alumno en ella. Puede ser que en secretaría quieran meter los datos de las asignaturas antes de empezar el proceso de matrícula:

No pueden. Una solución sería incluirlos con los datos de los alumnos vacíos (nulos), lo cual no sería nada aconsejable. Los valores nulos se deben evitar siempre que sea posible.

Por tanto, hay que distinguir entre el tipo de entidad Alumnos y el tipo de entidad Asignaturas. Ambas se relacionarán mediante un tipo de relación Matrícula. Los restantes tipos de entidad serán: Profesores y Aulas.

Los atributos de cada tipo de entidad:

- Alumnos: DNI, Apellidos y Nombre, Domicilio, Teléfono y COU
- Asignaturas: Código, Título, Créditos
- Profesores: DNI, Apellidos y nombre, Domicilio y Teléfono
- Aulas: Edificio y Número

Aún nos falta un atributo, que es la **nota**: ¿Dónde se coloca? En Alumnos no porque un alumno tiene muchas notas, tantas como asignaturas en las que esté matriculado. En Asignaturas no porque en la misma asignatura están matriculados muchos alumnos. Va a ser un atributo del tipo de relación **matrícula**.

Elección de los tipos de relación

El primer tipo de relación es Matrícula que relaciona cada alumno con las asignaturas en las que está matriculado. Además, esta relación tiene un atributo, nota, que se asocia cada tupla de la relación. El segundo tipo de relación es Supervisa que va de Profesores a Profesores y que incluye los papeles Supervisor y Supervisado. La última es Imparte, que relaciona cada profesor con la asignatura que imparte y el aula en la que da esa asignatura. Aquí también surgen varias posibilidades:

- a) Hacer dos relaciones binarias. Por ejemplo, profesor con asignatura y asignatura con aula.
- b) Hacer una relación ternaria entre profesores, aulas y asignaturas.
- c) Hacer tres relaciones binarias Profesores-Asignaturas, Profesores-Aulas, Asignaturas-Aulas.

Diferencias:

1. En las opciones a) y c) se permite que un profesor imparta una asignatura que aún no tiene aula (esto puede ser deseable o no).

2. El problema de a) es:

Profesores-Asignaturas = { ({DNI=6666666, NombreYApe=Rómulo Melón},{Cod=MD,...}) ... }

Asignaturas-Aulas = { ({Cod=MD,...},{Edif=Mates, NumAula=S103}), ({Cod=MD,...},{Edif=Biológicas, =104}) }

Estas relaciones son posibles, hay más de un curso de primero y por eso la misma asignatura se imparte en varias aulas. Ahora bien, Don Rómulo quiere saber en qué aula debe dar su clase de Matemática discreta, y para ello pregunta en secretaría. ¿Qué se le contesta?

Sin embargo, con la propuesta b) sí se le puede asignar a cada profesor un aula concreta para cada una de sus asignaturas.

El problema de c) sigue siendo el mismo:

Profesores-asignaturas = { ({DNI=6666666, NombreYApe=Rómulo Melón,...},{Cod=MD,...}), ({DNI=6666666, NombreYApe=Rómulo Melón,...},{Cod=IS,...}), ... }

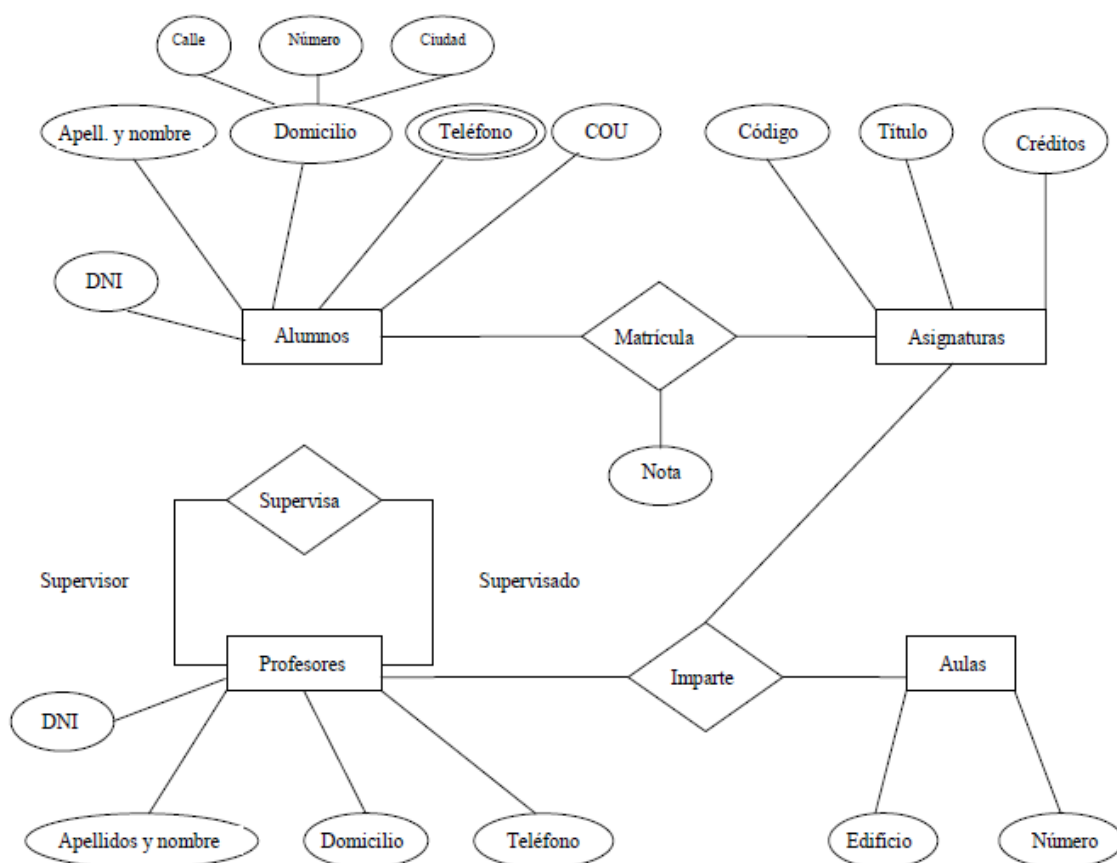
Asignaturas-Aulas = { ({Cod=MD,...}, {Edif=Mates, NumAula=S103}), ({Cod=MD,...}, {Edif=Biológicas, =104}), ({Cod=IS,...}, {Edif=Mates, NumAula=S103}), ({Cod=IS,...}, {Edif=Biológicas, NumAula=104}) }

Profesores-Aulas = {{DNI=6666666, NombreYApe=Rómulo Melón,...},
 {Edif=Mates, NumAula=S103}}, {{DNI=6666666, NombreYApe=Rómulo Melón,...},
 {Edif=Biológicas, NumAula=104}}, ...}

Don Rómulo sabe que da 2 asignaturas, cada una en un aula, pero sigue sin saber a dónde tiene que ir a dar MD.

Conclusión: Una relación ternaria tiene en general más información que 3 binarias.

El diagrama entidad-relación del ejemplo quedaría como se ilustra a continuación:



Adelanto de las restricciones de integridad

Con los elementos anteriores tenemos una primera aproximación a los diagramas E-R en la que tenemos definidos los elementos principales de los diagramas. Sin embargo, en el modelo E-R también se pueden definir numerosas restricciones de integridad sobre los tipos de entidades y tipos de relaciones.

Por ejemplo, en la relación Supervisa un profesor puede tener a lo sumo un supervisor, pero el diagrama anterior permite

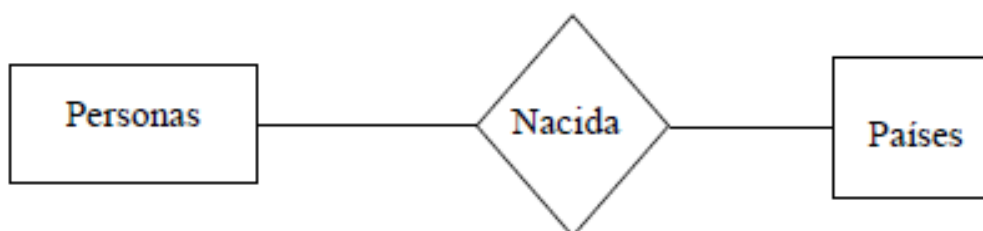
SUPERVISOR	SUPERVISADO
{DNI=666666,...},	{DNI=444444,...}}
{DNI=000001,...},	{DNI=444444,...}}

Que no debería ser una instancia válida de la relación.

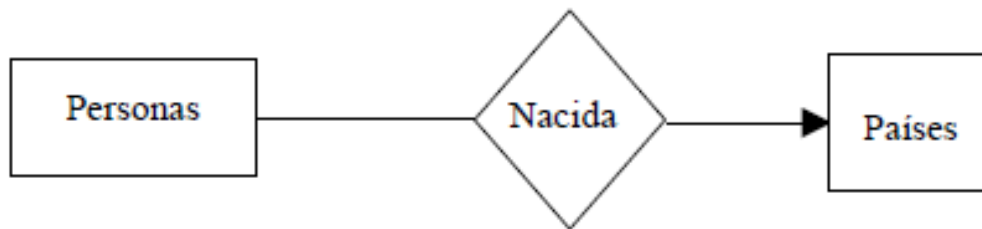
Para evitar estas situaciones se introducen las denominadas restricciones de integridad de la base de datos. Las restricciones de

integridad (o simplemente restricciones) son propiedades que se asocian a un tipo de entidad o de relación. Las instancias válidas del tipo de entidad o relación son aquellas en las que se verifique el conjunto de restricciones asociadas. Las restricciones son parte del diseño de la base de datos igual que los tipos de entidades o de relaciones. Los SGBD se encargan de comprobar que la instancia verifica las restricciones más usuales. En el ejemplo anterior, una vez incluida la restricción, el SGBD no nos permitiría insertar la segunda fila.

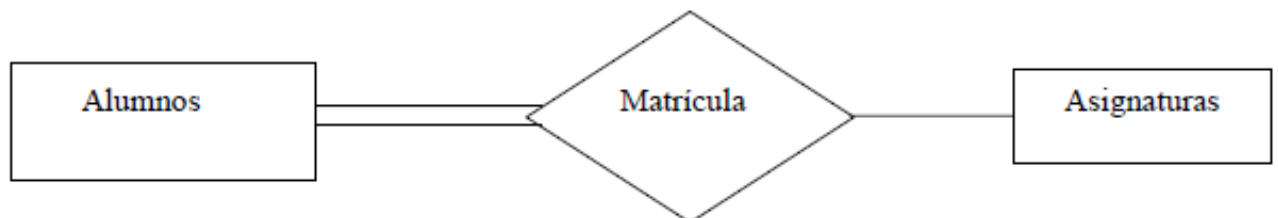
Un tipo de restricción de integridad que interesa conocer en esta etapa es la restricción de clave. Una restricción de clave consiste en imponer que un conjunto de atributos sea el que defina unívocamente a una fila de un tipo de entidades. Por ejemplo, en el tipo de entidades Alumnos se puede elegir DNI para identificar a un alumno en concreto, pero no sería conveniente usar el atributo Nombre y apellidos porque es muy posible encontrar a dos personas con los mismos nombres y apellidos. Por motivos de eficiencia conviene que el número de atributos elegidos sea el menor posible. A veces, es posible elegir varios conjuntos de atributos que contengan el mismo número de atributos, pero se suele escoger uno de estos conjuntos como el representativo, que se denomina clave primaria. Por ejemplo, si hubiese un atributo Número de la Seguridad Social, se podría usar también como clave. Todos estos conjuntos con el menor y mínimo número de atributos se denominan colectivamente como claves candidatas. Otro tipo de restricción de integridad importante ahora son las restricciones de cardinalidad. La idea de este tipo de restricción se puede entender con el siguiente ejemplo: supongamos que deseamos tener información sobre el país de nacimiento de personas. Habría una relación Nacida entre las entidades Personas y Países, como se muestra a continuación:



Además, deseamos expresar que, si bien muchas personas pueden haber nacido en un país, una persona en concreto sólo puede haber nacido en un país. Esto se expresa en el diagrama E-R con una flecha que indica que una persona ha nacido en un país en concreto. Leyendo la relación en el sentido contrario diríamos que en un país pueden haber nacido muchas personas (el segmento que une Nacida con personas no lleva flecha). (Esta restricción de cardinalidad también la podemos encontrar en el ejemplo de la secretaría, en la relación Supervisa, como se ha visto en el inicio de este apartado).



El último tipo de restricción de integridad que interesa introducir ahora es la participación total. Se refiere a que podamos encontrar cada entidad de un tipo de entidad en la relación que lo liga con otro u otros. Por ejemplo, en la base de datos de la secretaría, si hay un alumno en concreto dado de alta en la base de datos es porque se debe haber matriculado de alguna asignatura. Es decir, cada alumno definido en el tipo de entidad Alumnos debemos encontrarlo en la relación Matrícula, relacionado con la asignatura en la que esté matriculado. En el diagrama E-R se expresa con una línea doble, como se ve a continuación:



3. Diseño lógico

El diseño lógico es la segunda etapa del diseño de bases de datos en general y de las bases de datos relacionales en particular. En nuestro caso, las BD relacionales, el resultado de esta etapa es un esquema relacional basado en un modelo relacional. En este apartado se describirá en primer lugar el modelo relacional y en segundo lugar cómo pasar de un esquema entidad-relación a un esquema relacional.

El modelo relacional

Este modelo fue creado por Codd a principios de los 70 al que dotó de una sólida base teórica. Actualmente está implementado en la mayoría de los SGBD usados en la empresa. El concepto principal de este modelo es la relación o tabla. Es importante no confundir la tabla con las relaciones del modelo E-R. Aquí las relaciones se aplican tanto a tipos de relaciones como a tipos de entidades. En este modelo no se distingue

entre tipos de entidades y tipos de relaciones porque la idea es que una relación o tabla expresa la relación entre los tipos de valores que contiene.

A continuación se introducen los conceptos de este modelo:

- Entidad. Igual que en el modelo E-R. También se les llama tuplas o filas de la relación.
- Atributo. Igual que en el modelo E-R. También se le llaman campos o columnas de la relación. El dominio de los atributos tiene que ser simple: no se admiten atributos multivalorados ni compuestos.
- Esquema de una relación. Viene dado por el nombre de la relación y una lista de atributos. Es el tipo de entidad.
- Conjunto de entidades. Relación o tabla.

Por ejemplo, el tipo de entidad Alumnos del modelo E-R del apartado del diseño conceptual se representaría como la siguiente relación:

Alumnos (DNI, NombreYApellidos, Domicilio, Teléfono, COU)

El orden de los atributos en la lista no importa. Lo fijamos porque nos viene bien para representarlo como tabla, pero cualquier permutación es válida.

- Clave. Igual que en el modelo E-R. Hay que darse cuenta que en el modelo relacional todas las tablas deben tener claves, y que algunas tablas van a representar relaciones del esquema E-R.
- Instancia de una relación. Son conjuntos de entidades. Cada entidad se representa como una tupla. Cada componente de la tupla corresponde con el valor del atributo correspondiente, según el orden enunciado en el esquema de la relación.

Por ejemplo, una instancia de la relación Alumnos sería:

{ (01234567Z, Manuel Vázquez Prieto, Calle del Jazmín 7 4 Izq, 91-12345678, COU = Sí),}

En el modelo relacional no se representan diagramas del esquema de la BD. Por el contrario, el esquema relacional se representa por los conjuntos de entidades como hemos visto antes (nombre de la tabla y entre paréntesis el nombre de sus atributos). Las instancias de una relación se representan con tablas, como se muestra en el ejemplo del conjunto de entidades Alumnos:

<i>DNI</i>	<i>NombreYApellidos</i>	<i>Domicilio</i>	<i>Teléfono</i>	<i>COU</i>
<i>01234567Z</i>	<i>Manuel Vázquez Prieto</i>	<i>Calle del Jazmín 7 4 Izq</i>	<i>9112345678</i>	<i>SI</i>
...				

Paso de un esquema E-R a un esquema relacional

A continuación se describe la forma de traducir cada uno de los elementos que aparecen en el modelo E-R a un esquema relacional.

Tipos de entidades

Para cada tipo de entidad se crea una relación con el mismo nombre y conjunto de atributos. Por ejemplo, en el caso de la BD de secretaría los tipos de entidades dan lugar a las siguientes relaciones:

Alumnos(DNI, Apellidos y Nombre, Domicilio, Teléfono, COU)

Asignaturas(Código, Título, Créditos)

Profesores(DNI, Apellidos y nombre, Domicilio, Teléfono)

Aulas(Edificio, Número)

Tipos de relaciones

Para cada tipo de relación R se crea una relación que tiene como atributos:

Los atributos de la clave primaria de cada tipo de entidad que participa en la relación.

Los atributos de la propia relación.

En ocasiones hay que renombrar atributos para evitar tener varios con el mismo nombre.

Como ejemplo, en el caso de la BD de secretaría los tipos de relación dan lugar a las siguientes relaciones:

Matrícula(DNI, Código, Nota)

Supervisa(DNISupervisor, DNISupervisado)

Imparte(DNI, Código, Edificio, NumAula)

Obsérvese que en la relación Supervisa se han renombrado los atributos DNI para indicar el papel de cada uno de ellos en la relación y además evitar que se use el mismo nombre para más de un atributo.

Claves

Hay dos casos:

1. La relación proviene de un tipo de entidad en el esquema E-R. La clave es la misma que la del tipo de entidad.

Por ejemplo:

Alumnos(DNI, Apellidos y nombre, Domicilio, Teléfono, COU)

2. La relación proviene de un tipo de relación en el esquema E-R. Si la relación R es un tipo de relación entre varios tipos de entidades se va a construir una relación bajo el modelo E-R a partir de R con los atributos que forman clave primaria en todas las entidades participantes más los propios de R. De ellos formarán clave primaria las claves primarias de cada una de las entidades participantes.

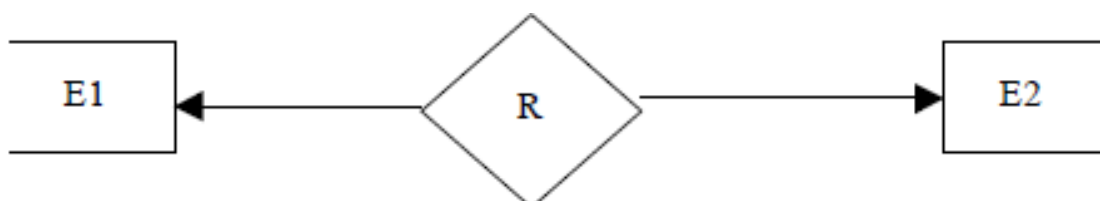
Por ejemplo:

Matrícula(DNI, Código, Nota)

Restricciones de cardinalidad

Es posible incorporar este tipo de restricciones de integridad cuando se desean indicar relaciones una a una, una a varias y varias a varias. (En el ejemplo de la secretaría tenemos la relación Supervisa, del tipo una a varias). A continuación se muestran estos casos para relaciones binarias, siendo c1 y c2 las claves primarias de E1 y E2, respectivamente:

a) Una a una



De los atributos de la traducción de R al modelo relacional podemos escoger como clave o bien c1 o bien c2, de ellas la más adecuada será la que tenga menos atributos. Esto es posible porque cada entidad de E1 está relacionada con sólo una de E2 y viceversa, por lo que no es posible que la misma entidad de E1 o de E2 aparezca más de una vez en R. Por tanto, cualquiera de sus claves primarias puede ser clave de R.

Por ejemplo, supongamos que:

Hombres(DNI)={1,2}, Mujeres(DNI)={3,4}, la relación Casado(DNIH, DNIM) sólo puede contener {(1,3), (2,4)} o {(1,4), (2,3)} o, en representación tabular:

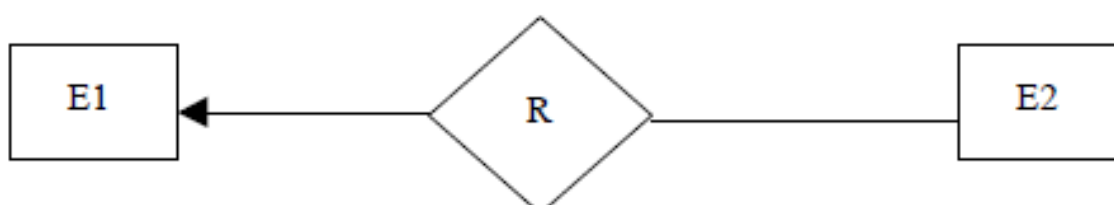
Hombres	Mujeres	Casados	Casados
DNI	DNI	DNIH	DNIM
1	3	1	3
2	4	2	4

Casados	Casados
DNIH	DNIM
1	4
2	3

Dado que no se repite ningún valor de las columnas de las dos posibilidades de R, tanto A como B podrían ser clave. Así, tendríamos las alternativas siguientes para la relación

Casados: Casados(DNIH, DNIM) y Casados(DNIM, DNIH).

b) Una a varias



En este caso, la clave de R debe ser la clave primaria de c2. Es decir, en la relación R no puede aparecer repetido ningún valor de E2.

En el ejemplo de las personas nacidas en países, tendríamos una instancia de la relación Nacida:

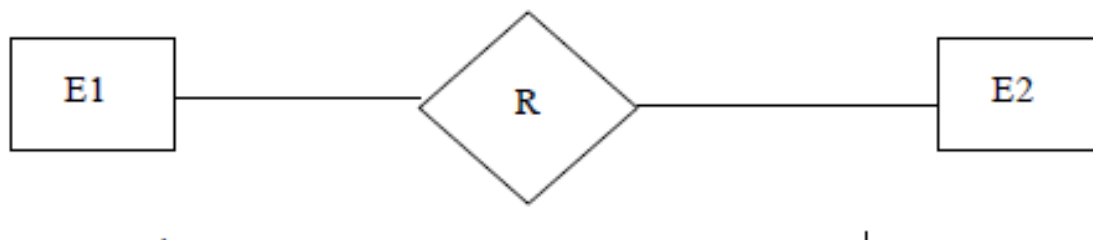
IDPersona	País
1	España
2	España
3	Portugal

En donde vemos que los valores de los países se pueden repetir en la relación, pero no el identificador de la persona porque, si así

fuese, significaría que la misma persona ha nacido en diferentes países. Otro ejemplo es el de la secretaria, en el que la relación

Supervisa quedaría $\text{Supervisa}(\text{DNISupervisor}, \underline{\text{DNISupervisado}})$.

c) Varias a varias



Éste es el caso más general en el que no se puede imponer ninguna restricción además de la ya indicada de clave. Por ejemplo,

Los tres casos anteriores se referían a relaciones binarias. En el caso de que se trate de relaciones n-arias, supongamos que la relación proviene de un tipo de relación R entre tipos de entidad E1, E2, ..., Ek, entonces:

- Si todos participan con cardinalidad varios en R, entonces una clave es la unión de las claves de E1, E2, ..., Ek.
- Si algunos tipos de entidad participan con cardinalidad una en R, entonces uno de ellos puede ser excluido de la superclave.

Restricciones de integridad

Hemos visto cómo se traducen las claves de los tipos de entidades y cómo aparecen claves en la traducción de los tipos de relaciones. Sin embargo, no es el único tipo de restricciones de integridad que aparece automáticamente al traducir un esquema E-R en otro relacional. Hay dos: **restricciones de integridad referencial** y **restricciones de participación total**.

Las claves y las restricciones de integridad referencial son características que se expresan directamente en la práctica totalidad de los SGBD relacionales. Estos sistemas se ocupan automáticamente de que no se violen estas restricciones. Sin embargo, no ocurre lo mismo con las de participación total y otras restricciones, como se verá en el tema dedicado a las restricciones de integridad.

Restricciones de integridad referencial

Al traducir un tipo de relación R, en cualquier instancia de R se debe cumplir que los valores de los atributos que hereda de una entidad (de su clave primaria) deben aparecer previamente en el conjunto de entidades. En el ejemplo de los hombres y mujeres casados está claro que en la relación Casados no puede aparecer un valor del DNI de un hombre o de una mujer que no estén previamente en el conjunto de entidades Hombres o Mujeres.

Es decir:

Hombres	Mujeres	Casados	
DNI	DNI	DNIH	DNIM
1	3	1	5
2	4	2	4

¡Incorrecto!

En esta instancia se estaría dando la idea de que el hombre con DNI 1 está casado con la mujer con DNI 5. Pero no sabemos nada de esta mujer dado que no está en el conjunto de entidades Mujeres (hay que considerar que este tipo de entidad tendría otros muchos atributos, como el nombre y apellidos de la mujer, su domicilio, etc., que podrían ser útiles para la aplicación de la base de datos). No obstante, lo que sí es posible que algunos hombres o algunas mujeres no estén casados entre sí.

Es necesario, por tanto imponer una restricción de integridad referencial entre los atributos clave heredados de una entidad con las clave de esa entidad. En este ejemplo, se podría expresar:

$$\text{Casados.DNIH} \subseteq \text{Hombres.DNI}$$

$$\text{Casados.DNIM} \subseteq \text{Mujeres.DNI}$$

Es decir, los valores de DNI que aparecen en el atributo DNIH de Casados deben aparecer previamente en el atributo DNI de Hombres (y lo análogo para las mujeres).

Restricciones de participación total

Cuando cada valor de un tipo de entidad debe aparecer en un tipo de relación, como Alumnos en Matrícula, significa que, además de la restricción de integridad referencial comentada en el apartado anterior, se debe cumplir que todo valor de DNI en Alumnos debe aparecer en el atributo DNI de Matrícula. Esto se puede expresar:

$$\text{Alumnos.DNI} \subseteq \text{Matrícula.DNI}$$

Por otra parte, dado que la restricción de integridad referencial sobre esta tabla arroja:

$$\text{Matrícula.DNI} \subseteq \text{Alumnos.DNI}$$

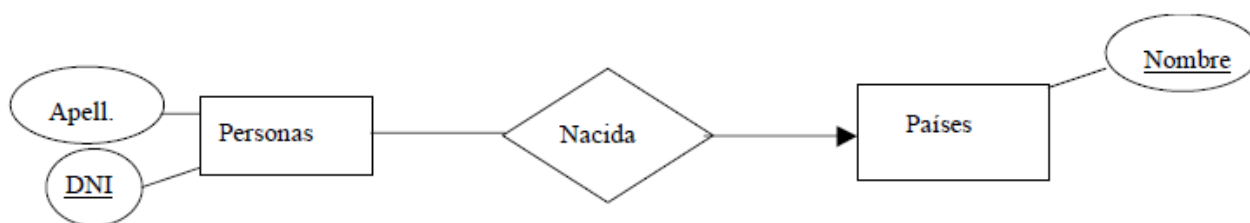
Llegamos a la conclusión de que:

$$\text{Matrícula.DNI} = \text{Alumnos.DNI}$$

Es decir, si aparece un valor de DNI en Matrícula, también debe aparecer en el atributo DNI de Alumnos, y viceversa.

Cuestiones de diseño

En ocasiones es posible combinar dos o más tablas en una sola. Generalmente se combinan por motivos de rendimiento. Por ejemplo, dado el ejemplo de personas nacidas en países:



La traducción de este esquema E-R al relacional sería:

Personas(DNI, Apell.)

Países(Nombre)

Nacida(DNI, Nombre)

Y se podrían combinar las dos primeras tablas en el nuevo esquema:

Personas(DNI, Apell, PaísNac)

Países(Nombre)

Un inconveniente de esta combinación es que, dado que no se exige participación total de Personas en Nacida, no tendremos información del país de nacimiento de algunas personas, y en la tabla Personas va a aparecer un valor NULL (nulo) en el atributo PaísNac, que indica que no se dispone de esa información.

El valor NULL es un valor que puede contener cualquier atributo, y lo soportan todos los SGBD. Es un valor especial que se debe tratar con cuidado y, en general, evitar, porque puede representar muchas cosas, tales como:

- Ausencia de información.
- Este atributo no se aplica o no tiene sentido para esta entidad en concreto.
- Valor desconocido.

Además causan problemas a la hora de realizar consultas sobre la base de datos. Por otra parte, ningún atributo que forme parte de una clave puede tomar el valor NULL.

4. Diseño físico

El objetivo del diseño físico es la generación del esquema físico de la base de datos en el modelo de datos que implementa el SGBD. Esto incluye la definición sobre el SGBD de las tablas con sus campos, la imposición de todas las restricciones de integridad y la definición de índices.

Los índices son estructuras de datos implementadas con ficheros que permiten un acceso más eficaz a los datos. Se organizan con respecto a uno o más campos (los denominados campos clave del índice, que no hay que confundir con el concepto de clave del modelo entidad-relación y relacional) y guardan sólo la información del valor de la clave y la dirección física a partir de la cual se pueden encontrar registros con ese valor.

Los índices son secuencias de registros que tienen dos campos: el valor de la clave y la dirección física del registro del fichero de datos en donde se puede encontrar una tupla con ese valor.

El rendimiento (el tiempo que se tarda en realizar una determinada operación) de una base de datos depende fundamentalmente de las operaciones de lectura y escritura en disco. Como las tablas generalmente no caben todas en la memoria principal, en general es necesario leer los datos del disco. Cuantos menos datos se lean o escriban en disco mejor será el rendimiento. Los índices permiten disminuir el tiempo de entrada/salida a disco.

Internamente, cuando el SGBD necesita buscar un registro según un valor de un campo (por ejemplo, un número de DNI) sobre el que se ha definido un índice para resolver alguna consulta, busca el valor en el índice, consulta la dirección del registro adjunto y a continuación busca en el fichero de datos (donde se almacenan los datos de la tabla correspondiente) el registro.

Esto es más rápido que hacer una búsqueda secuencial en el fichero de datos. Por un lado porque los índices no requieren en general una exploración secuencial de sus registros hasta encontrar el valor deseado, sino que se organizan como estructuras que permiten localizar el valor en menos tiempo. Por otro lado, cuando se recorre el índice se hace sobre registros pequeños, en comparación con los registros más grandes que contiene el fichero de datos; por lo tanto, el número de accesos a disco es menor.

No obstante, todo tiene un precio. Si se declara un índice, ese índice se debe mantener actualizado cada vez que la tabla sufra cualquier modificación. Si se definen muchos índices, el rendimiento de las actualizaciones se puede reducir significativamente. Por otra parte, si hay alternativas, siempre es mejor definir índices para los campos de menor tamaño, ya que cuanto más pequeño sea el campo clave, más pequeño será el índice y se necesitarán menos operaciones de lectura del índice.

Los índices se pueden definir como únicos o no (es decir, con duplicados o sin ellos). Los índices únicos indican que se aplican sobre campos en los cuales no debe haber elementos repetidos. Todas las claves primarias llevan asociado un índice de forma predeterminada. También se puede indicar que acepten valores nulos o no. Si se aceptan, el índice permitirá esos valores nulos, pero los registros que los contengan no estarán apuntados por el índice.

Restricciones de integridad

En este tema se trata uno de los aspectos más importantes para añadir consistencia a los diseños de bases de datos: son las restricciones de integridad que ayudan a mantener la consistencia semántica de los datos. Además de las restricciones de integridad definidas por las claves, las restricciones de cardinalidad y

las de participación total estudiadas anteriormente, se tratan las restricciones de los dominios, la integridad referencial, las dependencias funcionales y las dependencias multivaloradas.

Las restricciones de integridad proporcionan un medio de asegurar que las modificaciones hechas a la base de datos por los usuarios autorizados no provoquen la pérdida de la consistencia de los datos. Protegen a la base de datos contra los daños accidentales (no contra daños intencionados, de lo cual se ocupa la seguridad de las bases de datos). Los tipos de restricciones de integridad en una base de datos se pueden resumir como sigue:

- Claves.
- Cardinalidad de la relación.
- Restricciones de los dominios.
- Integridad referencial.
- Participación total.
- Dependencias funcionales.
- Otras restricciones.

En este último caso se recogen las restricciones que no se pueden catalogar en los casos anteriores. Por ejemplo, si se tiene una tabla en la que se describan un compuesto químico y sus componentes, sería deseable imponer que la suma de los componentes, expresados en porcentajes, fuese el 100 por cien.

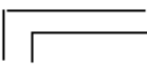
En las bases de datos tenemos los siguientes mecanismos para implementar las restricciones de integridad:

- Declaración de claves.
- Declaración de integridad referencial.
- Declaración de tipo de cardinalidad.
- Disparadores (Triggers).

Restricciones de los dominios

Las restricciones de los dominios son la forma más simple de restricción de integridad. Se especifica para cada atributo un dominio de valores posibles. Una definición adecuada de las restricciones de los dominios no sólo permite verificar los valores introducidos en la base de datos sino también examinar las consultas para asegurarse de que tengan sentido las comparaciones que hagan. Por ejemplo, normalmente no se considerará que la consulta "Hallar todos los clientes que tengan el nombre de una sucursal" tenga sentido. Por tanto, nombre-cliente y nombre-sucursal deben tener dominios diferentes.

La cláusula **check** de SQL:1999 permite restringir los dominios de maneras poderosas que no permiten la mayor parte de los sistemas de tipos de los lenguajes de programación. La cláusula check permite especificar un predicado que debe satisfacer cualquier valor asignado a una variable cuyo tipo sea el dominio. Por ejemplo:


 Número de cifras
 Número de decimales

create domain *sueldo-por-hora* **numeric**(4,2)
constraint *comprobación-valor-sueldo* **check**(value ≥ 6.00)

Restricciones de existencia

Dentro de las restricciones de los dominios, un tipo especial de restricción que se puede aplicar a cualquier dominio es la restricción de existencia. Esta restricción evita la aparición de valores nulos en las columnas. Se especifica indicando en la creación de la tabla cuáles son los atributos que no pueden contener valores nulos. De manera predeterminada, los atributos que formen parte de la clave primaria tienen esta restricción impuesta. Para declarar esta restricción en la definición de la tabla se usaría NOT NULL después del nombre del atributo y su dominio.

Restricciones de unicidad

Otro tipo especial de restricción que se puede aplicar a cualquier dominio es la restricción de unicidad. Esta restricción evita la aparición de valores duplicados en las columnas (de forma parecida a lo que hace la clave primaria). Por ejemplo:

Sólo se admite una sucursal en cada ciudad.

```
CREATE TABLE Sucursales
(nombre-sucursal VARCHAR(20) ,
ciudad-sucursal VARCHAR(20) NOT NULL,    - Restricción de existencia
PRIMARY KEY(nombre-sucursal)
UNIQUE (ciudad-sucursal))                - Restricción de unicidad
```

Restricciones de integridad referencial

La integridad referencial permite asegurar que un valor que aparece en una relación para un conjunto de atributos determinado aparezca también en otra relación para ese mismo conjunto de atributos.

Este tipo de restricciones se denota simplíficadamente:

$$\text{Relación1.}(\text{Atributo1-1}, \dots, \text{Atributo1-N}) \subseteq$$

$$\text{Relación2.}(\text{Atributo2-1}, \dots, \text{Atributo2-N})$$

El conjunto de atributos {Atributo1-1,...,Atributo1-N} se denomina clave externa, mientras que el conjunto de atributos {Atributo2- 1,...,Atributo2-N} es la clave primaria de Relación2.

Ya se ha visto cómo aparece este tipo de restricción de integridad en la traducción del esquema E-R al relacional. El sistema, por su parte, puede asegurar la imposición de estas restricciones de integridad, evitando la aparición de valores que las violen.

En concreto, la modificación de la base de datos puede ocasionar violaciones de la integridad referencial. Se distinguen tres casos:

- Al insertar una tupla es necesario comprobar que haya otra con los valores de la clave externa igual a los de sus atributos clave.
- Al borrar una tupla de R el sistema debe calcular el conjunto de tuplas de las otras relaciones que hacen referencia a R. Si este conjunto no es el conjunto vacío, o bien se rechaza la orden borrar como error, o bien se deben borrar las todas las tuplas que hacen referencia a R. La última solución puede llevar a borrados en cascada, dado que las tuplas pueden hacer referencia a tuplas que hagan referencia a R, etcétera.
- Actualizar. Hay que considerar dos casos: las actualizaciones de la relación que realiza la referencia (r2) y las actualizaciones de la relación a la que se hace referencia (r1).

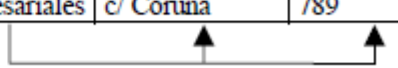
******Si se actualiza la tupla t2 de la relación r2 y esta actualización modifica valores de la clave externa α , se realiza una comprobación parecida al del caso de la inserción. El sistema debe asegurar que la nueva tupla encuentra sus valores de la clave externa en los de la clave primaria de r1.

******Si se actualiza la tupla t1 de la relación r1 y esta actualización modifica valores de la clave primaria, se realiza una comprobación parecida al del caso del borrado. Si quedan tuplas colgantes (sin correspondencia de clave externa con clave primaria), la actualización se rechaza como error o se ejecuta en cascada de manera parecida al borrado.

Dependencias funcionales

Una dependencia funcional (DF) es una propiedad semántica de un esquema de relación que impone el diseñador. Determina el valor de un conjunto de atributos a partir del valor de otro conjunto de atributos. Por ejemplo, en la siguiente relación se combinan los datos de los empleados, como su código de identificación y nombre, y de los centros a los que están adscritos, como la dirección y el teléfono.

<i>Empleados_Centros</i>							
<u>Id empleado</u>	<u>NombreE</u>	<u>DirecciónE</u>	<u>Puesto</u>	<u>Salario</u>	<u>Centro</u>	<u>DirecciónC</u>	<u>TeléfonoC</u>
123A	Ana Almansa	c/ Argentaes	Profesor	20.000	Informática	c/ Complutense	123
012D	David Díaz	c/ Daroca	Ayudante	10.000	Informática	c/ Complutense	123
789C	Carlos Crespo	c/ Cruz	Catedrático	30.000	Empresariales	c/ Coruña	789



En este ejemplo se muestra gráficamente que el valor del conjunto de campos DirecciónC y TeléfonoC depende del valor del campo Centro. En concreto, a un centro en particular le corresponden unívocamente una dirección y un teléfono. Es decir, cada vez que aparezca una fila con el valor Informática para Centro, siempre le corresponderá los mismos valores para los campos DirecciónC y TeléfonoC.

Se dice entonces que tanto DirecciónC como TeléfonoC son dependientes funcionalmente de Centro. Por cada fila con un mismo valor de Centro se repiten los valores DirecciónC y TeléfonoC, lo que implica una redundancia de valores no deseable que se estudiará en el tema dedicado a la normalización.

Las dependencias funcionales se denotan de la siguiente forma:

Conjunto de atributos que determinan → Conjunto de atributos determinados

En el ejemplo anterior: {Centro} → { DirecciónC, TeléfonoC }

Disparadores

Un disparador es un mecanismo que se puede usar para implementar restricciones de integridad no soportadas directamente por el SGBD. Un disparador es una orden que el sistema ejecuta de manera automática como efecto secundario de la modificación de la base de datos. Los disparadores son mecanismos útiles para implementar restricciones de integridad, alertar a los usuarios o para realizar de manera automática ciertas tareas cuando se cumplen determinadas condiciones.

Un ejemplo que se aplica a la imposición de restricciones de integridad sería la implementación de la detección de la violación de las dependencias funcionales.

Otro ejemplo sería enviar indicar cuándo se ha alcanzado un stock mínimo de un producto y que, por tanto, se debe reponer.

Un último ejemplo sería una aplicación bancaria en la que, en lugar de permitir saldos de cuenta negativos, un banco puede tratar los descubierto dejando a cero el saldo de las cuentas y creando un préstamo por el importe del descubierto. Este préstamo recibe un número de préstamo idéntico al número de cuenta que ha tenido el descubierto.

Normalización

La traducción del esquema conceptual al lógico no es única. Es necesario contar con una medida de la calidad de la agrupación de los atributos en relaciones. Como herramienta principal se usan las dependencias funcionales para agrupar los atributos en esquemas de relación, que se dice que se encuentran en una determinada forma normal. La forma normal de un esquema de relación determina su grado de calidad con respecto a reducir dos efectos no deseados: la redundancia de datos y las anomalías que produce esta redundancia. Es importante determinar en qué forma normal se encuentra un esquema de relación y el procedimiento, conocido como normalización, para descomponerlo en otros esquemas de relación que se encuentren en formas normales más exigentes.

Redundancia de datos

Un objetivo del diseño de bases de datos relacionales es agrupar atributos en relaciones de forma que se reduzca la redundancia de datos y así el espacio de almacenamiento necesario. Por ejemplo, los siguientes dos esquemas

Empleados(Id_empleado, NombreP, DirecciónP, Puesto, Salario, Centro)

Centros(NombreC, DirecciónC, Teléfono)

contienen la misma información que el siguiente:

Empleados_Centros(Id_empleado, NombreP, DirecciónP, Puesto, Salario, NombreC, DirecciónC, Teléfono)

<i>Empleados</i>					
<u>Id empleado</u>	<u>NombreE</u>	<u>DirecciónE</u>	<u>Puesto</u>	<u>Salario</u>	<u>Centro</u>
123A	Ana Almansa	c/ Argentales	Profesor	20.000	Informática
456B	Bernardo Botín	c/ Barcelona	Administrativo	15.000	Matemáticas
789C	Carlos Crespo	c/ Cruz	Catedrático	30.000	CC. Empresariales
012D	David Díaz	c/ Daroca	Ayudante	10.000	Informática

<i>Centros</i>		
<u>NombreC</u>	<u>DirecciónC</u>	<u>Teléfono</u>
Informática	Complutense	123
Matemáticas	Complutense	456
CC. Empresariales	Somosaguas	789

<i>Empleados_Centros</i>							
<u>Id empleado</u>	<u>NombreE</u>	<u>DirecciónE</u>	<u>Puesto</u>	<u>Salario</u>	<u>Centro</u>	<u>DirecciónC</u>	<u>Teléfono</u>
123A	Ana Almansa	c/ Argentales	Profesor	20.000	Informática	Complutense	123
456B	Bernardo Botín	c/ Barcelona	Administrativo	15.000	Matemáticas	Complutense	456
789C	Carlos Crespo	c/ Cruz	Catedrático	30.000	CC. Empresariales	Somosaguas	789
012D	David Díaz	c/ Daroca	Ayudante	10.000	Informática	Complutense	123

La relación **Empleados_Centros** presenta redundancia de datos porque se repite para cada empleado la información asociada al centro. Las relaciones con datos redundantes presentan diferentes anomalías de actualización: son las anomalías de inserción, borrado y modificación.

Anomalías de actualización

- Anomalías de inserción. Se produce en dos casos. En primer lugar, cuando se inserta una nueva fila sin respetar las dependencias funcionales. En el ejemplo anterior puede ocurrir si se añade una fila de un empleado adscrito a Informática y con un teléfono distinto de 123. En segundo lugar, la imposibilidad de añadir nuevos datos para el consecuente de la dependencia funcional sin que exista un antecedente para ella. En el ejemplo anterior no se puede dar de alta un centro a menos que exista un empleado destinado en él. Sería necesario dejar valores nulos en la clave (Id_empleado).
- Anomalías de modificación. Se produce cuando se modifican las columnas con datos redundantes de sólo un subconjunto de las filas con el mismo dato. En el ejemplo puede ocurrir cuando se modifica el teléfono de Informática sólo en la primera fila.
- Anomalías de eliminación. Se produce cuando se eliminan todas las filas en las que aparecen los datos redundantes por lo que se pierde los datos de la dependencia funcional. Si se elimina la segunda fila porque el empleado se da de baja, se pierden también los datos del centro.

Formas normales y normalización

La forma normal de una relación se refiere a la mejor forma normal que satisface un esquema de relación indicando así el grado hasta el que se ha normalizado. La indicación del grado de calidad de un esquema de relación se refiere en general en el contexto global del esquema de la base de datos relacional, es decir,

en el conjunto de todos los esquemas de relación de la base de datos. Dos propiedades que se deben cumplir para poder asegurarlo son:

- La propiedad de preservación de dependencias, que asegura que las dependencias funcionales originales se mantienen en algún esquema de relación después de la descomposición.
- La propiedad de la posibilidad de reproducir la información de la tabla antes de su descomposición a partir de las tablas resultado de ella.

Las formas normales más habituales, por orden ascendente de exigencia de las propiedades deseadas, son:

Primera (1FN)

Segunda (2FN)

Tercera (3FN)

Boyce/Codd (FNBC)

Hay otras formas normales más exigentes y que se refieren a otras propiedades deseables. Sin embargo, la utilidad práctica de estas formas normales es cuestionable cuando las restricciones en que se basan son difíciles de entender o identificar por los diseñadores y usuarios. Así, en la práctica se usa hasta la forma normal de Boyce/Codd, aunque en general, los diseños prácticos exigen al menos 3FN.

El proceso de asegurar una forma normal para un esquema se denomina normalización.

Primera forma normal

Actualmente se considera como parte de la definición formal de relación, porque establece que los dominios de los atributos sólo pueden ser atómicos, para evitar atributos multivalorados, compuestos y sus combinaciones. En definitiva evita las relaciones dentro de las relaciones.

Por ejemplo, si se asume que en la entidad Centros, un centro puede tener más de un teléfono, podríamos tener una representación como la siguiente.

<i>Centros</i>		
<u>NombreC</u>	<u>DirecciónC</u>	<u>Teléfonos</u>
Informática	Complutense	{123, 321, 213}
Matemáticas	Complutense	{456}
CC. Empresariales	Somosaguas	{789, 987}

Sin embargo, esto supondría el uso del atributo multivalorado Teléfonos. Hay tres posibilidades de representar la entidad para satisfacer la primera forma normal:

- Eliminar el atributo Teléfonos y crear una nueva relación que asocie en cada fila un centro con un teléfono. La clave de la primera relación debe formar parte de la clave de la segunda relación. Presenta como inconveniente añadir una nueva relación al esquema de la base de datos y redundancia. Presenta

anomalías cuando se borra un centro y no se borran los teléfonos asociados. La integridad referencial asegura evitar las anomalías.

<i>Centros</i>	
<u>NombreC</u>	<u>DirecciónC</u>
Informática	Complutense
Matemáticas	Complutense
CC. Empresariales	Somosaguas

<i>Teléfonos</i>	
<u>NombreC</u>	<u>Teléfono</u>
Informática	123
Informática	321
Informática	213
Matemáticas	456
CC. Empresariales	789
CC. Empresariales	987

- Ampliar la clave de la relación de manera que incluya al atributo multivalorado. Presenta como inconveniente añadir redundancia que provoca anomalías.

<i>Centros</i>		
<u>NombreC</u>	<u>DirecciónC</u>	<u>Teléfono</u>
Informática	Complutense	123
Informática	Complutense	321
Informática	Complutense	213
Matemáticas	Complutense	456
CC. Empresariales	Somosaguas	789
CC. Empresariales	Somosaguas	987

- Si se conoce la cardinalidad máxima del atributo multivalorado, se pueden crear tantas columnas como la cardinalidad máxima. Presenta como inconveniente el uso de valores Null.

<u>NombreC</u>	<u>DirecciónC</u>	<u>Teléfono1</u>	<u>Teléfono2</u>	<u>Teléfono3</u>
Informática	Complutense	123	321	213
Matemáticas	Complutense	456	Null	Null
CC. Empresariales	Somosaguas	789	987	Null

Si el atributo multivalorado es compuesto, como es el caso de representar varias direcciones para un empleado:

Empleados(Id_empleado, NombreE, {Direcciones(Calle, Ciudad, CódigoPostal)})

Esta relación se puede descomponer en dos:

Empleados(Id_empleado, NombreE)

DireccionesE(Id_empleado, Calle, Ciudad, CódigoPostal)

Este procedimiento de desanidamiento se puede aplicar recursivamente a cualquier relación con atributos multivalorados. Teniendo en cuenta que es necesario propagar la clave de la relación original a la clave de la nueva relación, que contiene además la clave que identifica unívocamente al atributo multivalorado.

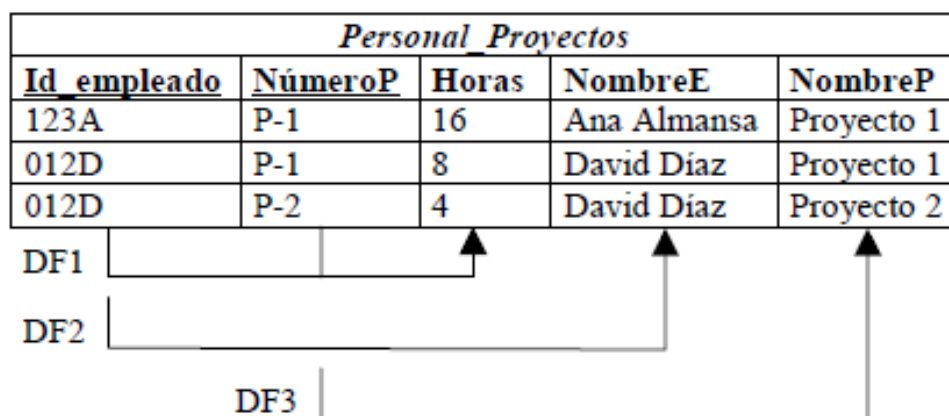
Segunda forma normal

En el ejemplo a continuación se puede observar que existen anomalías de actualización causadas por las dependencias funcionales DF2 y DF3, ya que como sus antecedentes no son clave, puede haber varias filas con el mismo consecuente. Se usa una notación gráfica para la expresión de las dependencias funcionales. Así:

$$DF1 = \{Id_empleado, NúmeroP\} \rightarrow \{Horas\}$$

$$DF2 = \{Id_empleado\} \rightarrow \{NombreE\}$$

$$DF3 = \{NúmeroP\} \rightarrow \{NombreP\}$$

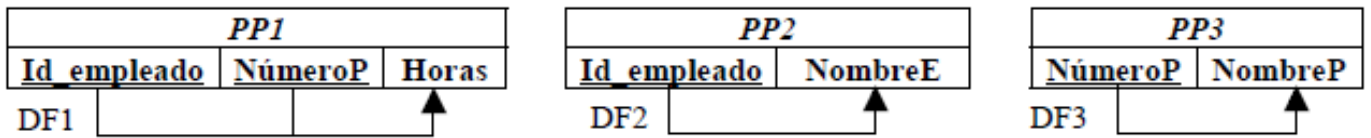


La segunda forma normal evita este tipo de anomalías.

Se dice que una relación está en segunda forma normal si cada atributo que no forme parte de la clave primaria depende funcional y completamente de cada clave.

Un esquema que no se encuentre en segunda forma normal puede traducirse en varios esquemas que sí lo estén. El procedimiento de normalización es crear tantas nuevas relaciones como dependencias funcionales no sean completas.

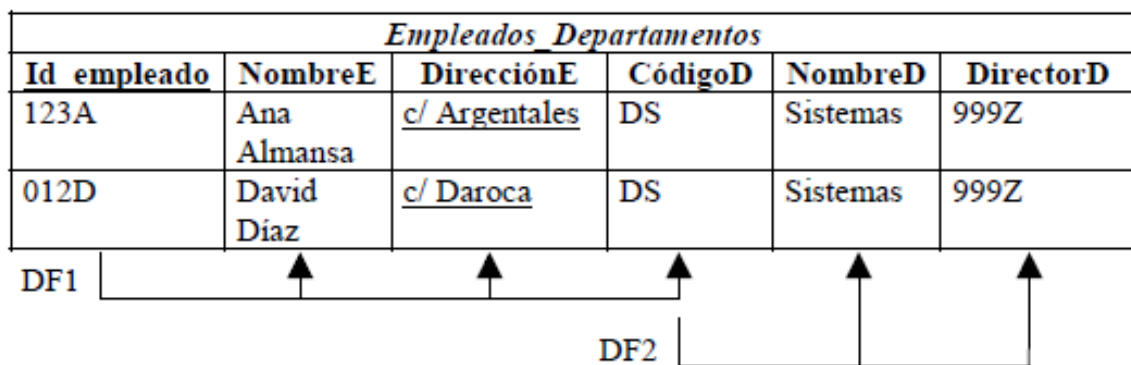
Así, el ejemplo anterior se traduce en:



Hay que observar que este procedimiento asegura que el resultado está, al menos, en segunda forma normal. En particular, y como se puede contrastar con la definición de otras formas normales, el resultado conseguido en este ejemplo se encuentra en FNBC, como se podrá comprobar más adelante.

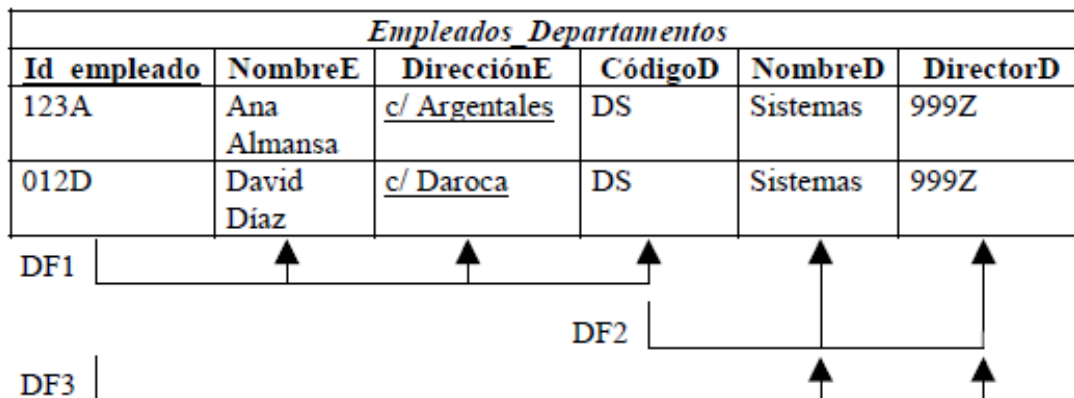
Tercera forma normal

En el ejemplo a continuación se puede observar que existen anomalías de actualización causadas por la dependencia funcional DF2. Sin embargo, este esquema está en segunda forma normal porque los dos últimos atributos, que son los que causan las anomalías, dependen completa (y transitivamente) del único atributo que forma la clave primaria.



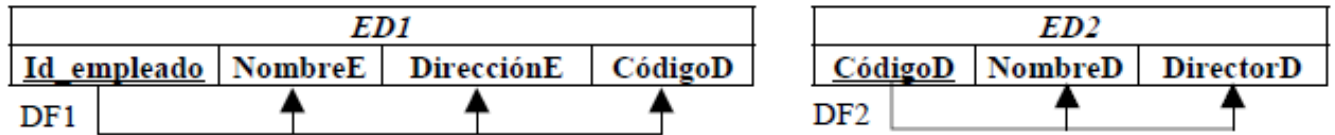
La tercera forma normal se basa en el concepto de dependencia funcional transitiva. Una dependencia funcional $X \rightarrow Y$ es una dependencia funcional transitiva si existe un conjunto de atributos Z que ni forman clave candidata ni son subconjunto de ninguna clave y además se cumple $X \rightarrow Z$ y $Z \rightarrow Y$.

En el ejemplo anterior, DF3 es una dependencia funcional transitiva:



Un esquema está en tercera forma normal si satisface la segunda forma normal y ninguno de los atributos que no forman parte de una clave candidata depende transitivamente de la clave primaria.

El procedimiento para normalizar esta relación consiste en descomponerla en los atributos definidos por la dependencia funcional responsable de la transitividad. En el ejemplo anterior, el resultado de la descomposición es:



El siguiente es un algoritmo que se puede aplicar para calcular estas descomposiciones, donde R es el esquema original a descomponer y F el conjunto de dependencias funcionales que se cumple en R:

D = {R}

1. Encontrar un recubrimiento mínimo T de F
2. for each $X \rightarrow Y \in T$, crear en D un esquema Ri con $\{X \cup \{A_1\} \cup \dots \cup \{A_k\}\}$
 si $R_i \not\subseteq R_j \in D$, dadas las D.F. $X \rightarrow \{A_1\}, \dots, X \rightarrow \{A_k\}$
3. Si ninguno de los esquemas contiene una clave de R, se crea uno nuevo.

Donde el recubrimiento mínimo es el resultado en primer lugar de eliminar todos los atributos de los antecedentes de las dependencias funcionales que sea posible y, en segundo lugar, todas las dependencias funcionales que sea posible.

Forma normal de Boyce-Codd

La forma normal de Boyce-Codd (FNBC) se propuso como una forma más simple que la tercera, pero en realidad es más estricta porque cada relación en FNBC está en 3FN pero lo contrario no se cumple.

Por ejemplo, considérese la siguiente relación, que está en 3FN pero no en FNBC. La FNBC evita redundancias que la 3FN no puede. En este ejemplo se almacena información de los investigadores participantes en proyectos, que pueden ser codirigidos, pero los investigadores principales no pueden dirigir más de uno.

<i>Proyectos</i>		
<u>Investigador</u>	<u>Proyecto</u>	<u>IPrincipal</u>
111A	Proyecto 1	123A
222B	Proyecto 1	012D
333C	Proyecto 1	123A
444D	Proyecto 2	789C
444D	Proyecto 1	123A

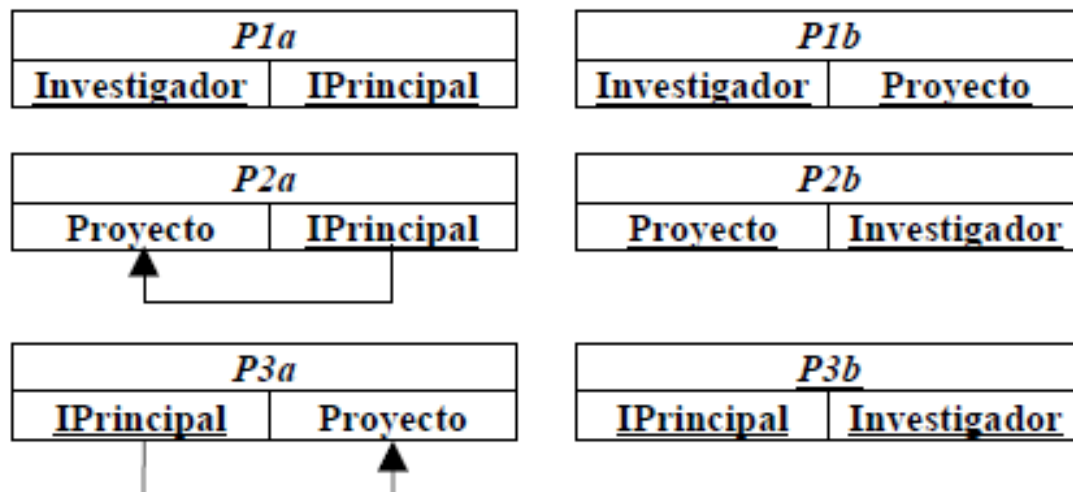
Functional dependencies are shown below the table:

DF1: A horizontal line from Investigador to Proyecto and IPrincipal, with an upward arrow pointing to the line.

DF2: A horizontal line from Proyecto to IPrincipal, with an upward arrow pointing to the line.

En este ejemplo, que cumple la 3NF, hay una anomalía que se debería poder evitar, porque si no se vigila la dependencia funcional DF2 se podría añadir una tupla de manera que una persona fuese investigadora principal de dos proyectos.

La descomposición del esquema no es inmediata, hay tres posibilidades:



Todas estas descomposiciones pierden la dependencia funcional DF1 porque las dependencias funcionales se refieren al contexto local de un esquema, no hacen referencia a esquemas externos. Aún peor, las dos primeras generan tuplas incorrectas cuando se trata de recuperar la información de la tabla original. Por ejemplo, en la primera descomposición se pierde la información de los investigadores principales de cada proyecto:

<i>P1a</i>		<i>P1b</i>	
<u>Investigador</u>	<u>IPrincipal</u>	<u>Investigador</u>	<u>Proyecto</u>
111A	123A	111A	Proyecto 1
222B	012D	222B	Proyecto 1
333C	123A	333C	Proyecto 1
444D	123A	444D	Proyecto 1
444D	789C	444D	Proyecto 2

<i>P1a ? P1b</i>		
<u>Investigador</u>	<u>Proyecto</u>	<u>IPrincipal</u>
111A	Proyecto 1	123A
222B	Proyecto 1	012D
333C	Proyecto 1	123A
444D	Proyecto 2	123A
444D	Proyecto 2	789C
444D	Proyecto 1	123A
444D	Proyecto 1	789C

Por tanto, ninguna de estas dos últimas descomposiciones es aceptable.

En la práctica, la mayoría de los esquemas que están en 3NF lo están también en FNBC.

Desnormalización para el rendimiento

A veces los diseñadores de bases de datos escogen un esquema que tiene información redundante; es decir, que no está normalizada. Utilizan la redundancia para mejorar el rendimiento para aplicaciones concretas. La penalización sufrida por no emplear un esquema normalizado es el trabajo extra (en términos de tiempo de codificación y de tiempo de ejecución) de mantener consistentes los datos redundantes.

Por ejemplo, supóngase que hay que mostrar el nombre del titular de una cuenta junto con el número y el saldo de su cuenta cada vez que se tiene acceso a la cuenta. En el esquema normalizado esto exige una reunión de cuenta con impositor.

Una alternativa para calcular la reunión sobre la marcha es almacenar una relación que contenga todos los atributos de cuenta y de impositor. Esto hace más rápida la visualización de la información de la cuenta. No obstante, la información del saldo de la cuenta se repite para cada persona titular de la cuenta, y la aplicación debe actualizar todas las copias, siempre que se actualice el saldo de la cuenta. El proceso de tomar un esquema normalizado y hacerlo no normalizado se denomina desnormalización, y los diseñadores lo utilizan para ajustar el rendimiento de los sistemas para dar soporte a las operaciones críticas en el tiempo.

Una alternativa mejor, soportada hoy en día por muchos sistemas de bases de datos, es emplear el esquema normalizado y, de manera adicional, almacenar la reunión en forma de vista materializada. (Una vista materializada es una vista de la base de datos cuyo resultado se almacena en la base de datos y se actualiza cuando se actualizan las relaciones utilizadas en la vista). Al igual que la desnormalización, el empleo de las vistas materializadas supone sobrecargas de espacio y de tiempo; sin embargo, presenta la ventaja de que conservar la vista actualizada es labor del sistema de bases de datos, no del programador de la aplicación.

Normativa de denominación

La normativa de denominación es una colección de reglas que permite asignar nombres a identificadores y objetos. El objetivo es que los nombres que se elijan indiquen de forma lo más clara posible el significado del elemento al que se refiere el nombre. Cada empresa suele usar una normativa diferente y más o menos complicada. A veces se tienen manuales de treinta páginas que la describen. En este documento se aconsejarán algunas reglas que se pueden usar para establecer una normativa de denominación.

Identificadores

Los identificadores (o nombres que se usan para designar los elementos de una base de datos) se construyen generalmente con letras y números. En muchos SGBD no se distinguen mayúsculas de minúsculas, pero su uso nos puede ayudar a hacer más legibles los identificadores. Cuando los identificadores tienen más de una palabra hay dos alternativas que se usan habitualmente:

- Separar cada palabra por un carácter de subrayado, como en Nombre_del_paciente.
- Separar cada palabra poniendo la primera letra de cada una en mayúsculas, como en

NombreDelPaciente.

Otra alternativa final, menos usada, es usar espacios en blanco para la separación, como en Nombre del paciente. Algunos SGBD tienen problemas con estos espacios en blanco y por eso no se usan habitualmente.

Incluso tienen problemas con algunos caracteres como las vocales acentuadas y las eñes, por lo que en general se evitan. En Access, sin embargo, no se tienen estos problemas y hemos seguido esta normativa de denominación de los identificadores.

Tablas

Las tablas representan entidades y sus nombres deberían describir las entidades que representan. Por ejemplo, Pacientes sería un identificador que describe a una tabla que contiene información sobre las entidades Pacientes. Además se escribe en plural porque el tipo de entidad contiene un conjunto de entidades (la tabla contiene varios pacientes en general).

Algunas tablas, sin embargo, no presentan entidades. Pueden representar relaciones entre entidades. Por ejemplo, la relación entre

Personas y Teléfonos se puede denominar Tiene teléfono. Cuando no se pueda encontrar un identificador adecuado para una relación siempre se puede escribir un identificador con los dos nombres de las tablas, como Personas/Teléfonos.

Reglas básicas de denominación de tablas:

- Seleccionar nombres de tablas basados en los nombres posibles para las entidades involucradas.
- Usar sustantivos para los nombres de las tablas.
- Asegurarse de que los nombres tienen un sentido intuitivo en la cultura de quienes utilizan la base de datos.
- Expresar las relaciones capturadas por las tablas mediante la vinculación de los nombres de las entidades vinculadas por la tabla.

En las tablas se tiene que denominar a las columnas. Las columnas representan elementos discretos de información sobre la entidad nombrada por la tabla. Normalmente tampoco representan relaciones. Debido a estos hechos el nombre de una columna debería ser un sustantivo que nombre el elemento de información que representa. Debería ser un sustantivo que reflejara la forma que los usuarios hablan sobre el elemento de información y debería reflejar características sobresalientes sobre el elemento de información.

Por ejemplo, se usará Nombre como identificador del atributo nombre de pila de un paciente.

Restricciones

Las restricciones se pueden denominar de formas autointerpretativas.

Hay que utilizar una abreviatura de dos letras para identificar la naturaleza de la restricción:

- CP (o PK en inglés, primary key) para clave principal
- IR (o RI en inglés, referential integrity) para integridad referencial
- CO (o CK en inglés, check) para la de comprobación
- UN para la de unicidad.

Después hay que utilizar el nombre de la tabla a la cual se aplica la restricción como segundo elemento del nombre. Una restricción de clave principal para la tabla Médicos se debería nombrar CP_Médicos.

En el caso de claves externas, donde están involucradas dos tablas, hay que poner el nombre de la segunda tabla como tercer elemento para el nombre de la restricción. Una clave externa para las tablas Médicos y Especialidades tendría el nombre IR_Médicos_Especialidades. Cuando están involucradas dos tablas hay que hacer que el segundo elemento del nombre sea la tabla con la clave externa y que el tercer elemento sea la tabla donde reside la clave principal.

Finalmente, para las restricciones de comprobación, que tienen un papel funcional, hay que agregar un elemento final al nombre que describe su función. Una restricción de comprobación que garantice el formato de un número telefónico, por ejemplo, podría tener este nombre:

CO_Médicos_FormatoCódigo.

Controles

Cada tipo de control se debería denominar con una indicación del tipo de control, anteponiendo a un nombre descriptor un prefijo que indique el tipo, como se propone en la siguiente tabla.

Control	Prefijo	Ejemplo
Cuadro de texto	txt (acrónimo de text, texto en inglés)	txtEntrada
Etiqueta	lbl (acrónimo de label, etiqueta en inglés)	lblEtiqueta
Cuadro combinado	cmb (acrónimo de combo box, cuadro combinado en inglés)	cmbProvincias
Botón de comando	but (acrónimo de button, botón en inglés)	butCancelar
Grupo de opciones (marcos)	fra (acrónimo de frame, marco en inglés)	fraOperaciones
Botón de opción	opt (acrónimo de option, opción en inglés)	optCubo
Botón de selección	chk (acrónimo de check, verificación en inglés)	chkVerde
Lista	lst (acrónimo de list, lista en inglés)	lstPacientes

Variables

Cada variable se debería denominar con una indicación del tipo de la variable, anteponiendo a un nombre descriptor un prefijo que indique el tipo, como se propone en la siguiente tabla.

Tipo de variable	Prefijo	Ejemplo
Byte	byt	bytEdad
Boolean	boo	booCasado

Objetos de la base de datos

Cada objeto de la base de datos se debería denominar con una indicación del tipo de objeto, anteponiendo a un nombre descriptor un prefijo que indique el tipo, como se propone en la siguiente tabla.

Tipo de objeto	Prefijo	Ejemplo
Tabla	tbl (acrónimo de table, tabla en inglés)	tblPacientes
Formulario	frm (acrónimo de form, formulario en inglés)	frmMédicos
Informe	rep (acrónimo de report, informe en inglés)	repPacientes
Consulta	qry (acrónimo de query, consulta en inglés)	qryGastoTrimestral