**BASES DE DATOS**

**¿Qué es un BD?**

El concepto de *base de datos*se refiere a una colección o depósito de datos integrados que debe tener dos características fundamentales: una redundancia controlada y una estructura que refleje las interrelaciones y restricciones existentes en el mundo real.  
Es esencial que los datos puedan compartirse por diferentes usuarios y entre distintas aplicaciones. Por tanto, los datos deben mantenerse independientes de las aplicaciones; y, además, la definición y la descripción que se haga de ellos —únicas para cada tipo de datos— han de estar almacenadas junto a dichos datos.

**Historia de las BD.**

1 **primera generación**: las bases de datos proceden de los sistemas de ficheros que se usaban antiguamente para cada aplicación. Antes de la década de 1960, lo más parecido a las bases de datos eran esos ficheros con información sin clasificar. Con frecuencia, una aplicación o un conjunto de aplicaciones, cada una con un fin concreto, usaban ficheros para guardar los datos. Una base de datos es un sistema formado por un conjunto de datos almacenados en discos, y es fundamental que se permita el acceso directo a ellos y el uso de un conjunto de programas para manipular esos datos. Sin embargo, al comenzar la década sí se empiezan a desarrollar los accesos remotos; por ejemplo, con el GUAM (Generalized Update Access Method), elaborado para el proyecto lunar Apollo. Además, se bajaron los precios de las computadoras para que las compañías privadas pudiesen adquirirlas, y de este modo popularizar el uso de los discos. Esto supuso un gran adelanto: a partir de este nuevo soporte se podía consultar directamente la información, **sin la necesidad de saber la ubicación exacta de los datos**. Y, como posibilitaba guardar estructuras de datos en listas y árboles, se iniciaron las primeras generaciones de bases de datos jerárquicas y de red.  
Otro de los principales logros de los años 60 fue la alianza entre 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.

2 **segunda** generación: A partir del trabajo de Edgar F. Codd, se desarrolló el Relational Software System, que actualmente se conoce como Oracle (el sistema de gestión de bases de datos relacional lleva el mismo nombre que la compañía: Oracle Corporation). Y esto condujo al desarrollo del SQL y la implementación de productos como DB2, SQL/DB y Oracle. Estos sistemas ya solventaban algunos problemas de las versiones anteriores, como el control de la redundancia de datos, y empezaron a independizar los tratamientos de los datos generando dos partes; esto es, diferenciando la parte del SGBD y de la BD. Se generaron estructuras de datos interrelacionados y se creó una estructura de datos integrada y centralizada. En la época de los 80 también se desarrollará el SQL (Structured Query Language). Se trata de un lenguaje de consultas o lenguaje declarativo de acceso a bases de datos relacionales. Permite efectuar consultas con el fin de recuperar información de interés de una base de datos y realizar cambios sobre ella de forma sencilla; además, permite analizar grandes cantidades de información y especificar diversos tipos de operaciones frente a la misma información, a diferencia de las bases de datos de los años 80 que se diseñaron para aplicaciones de procesamiento de transacciones. ompetencia del SQL Server, de la compañía Microsoft, y de otros sistemas administradores de bases de datos relacionales con licencia libre, como es el caso de PostgreSQL, MySQL o Firebird, que aparecieron después, en la década de 1990.

3 **tercera generación**: Las bases de datos de la anterior generación todavía son primitivas y, aunque empiezan a crecer, generan nuevos problemas; por ejemplo, los datos se van volviendo más complejos y las aplicaciones que los tratan requieren más datos. Además, la evolución en el hardware obliga a que se requieran nuevas implementaciones.  
Sin embargo, es a mediados de la década de los 80 cuando se modifica la primera publicación de ANSI del lenguaje SQL, cuando se comienzan a agregar nuevas expresiones regulares, consultas recursivas, *triggers* y algunas características orientadas a objetos. Y, posteriormente, ya en el siglo XXI, se volverá a modificar con la introducción de características de XML, cambios en sus funciones, estandarización del objeto *sequence* y de las columnas autonuméricas. Y, además, también en el siglo XXI, se creerá laposibilidad de que SQL se pueda utilizar junto con XML, y se definirán las maneras de cómo importar y guardar datos XML en una base de datos SQL. Esto facilitará a las aplicaciones la posibilidad de integrar el uso de XQuery (lenguaje de consulta XML) para el acceso concurrente a datos ordinarios SQL y documentos XML. Asimismo, se ofrecerá la posibilidad de usar la cláusula *Order by*. Como consecuencia de todos estos avances, se desarrollarán herramientas como Excel y Access, del paquete de Microsoft Office, que marcan el inicio de las bases de datos orientadas a objetos.

Un ODBMS (sistema de base de datos orientada a objetos) consigue que los objetos de la base de datos aparezcan como objetos de un lenguaje de programación en uno o más lenguajes de programación en los que se basa este sistema. Un ODBMS extiende los lenguajes con datos persistentes de forma transparente, control de concurrencia, recuperación de datos, consultas asociativas y otras capacidades.

Los ORDBMS (sistemas de bases de datos objetos-relaciones) nacen como una extensión del modelo relacional, extensión en la que los dominios de dicha base de datos ya no son solo atómicos. Por ello, no se cumple la primera forma normal, sino que las tuplas también pueden constituir una relación, algo que llevará a la creación de una relación de relaciones. De este modo, se abre la posibilidad de guardar objetos más complejos en una sola tabla, con referencias a otras relaciones. Y esto se acerca más al paradigma de la programación orientada a objetos o POO.

**Ventajas e inconveniente de las DB**

Sus principales inconvenientes son:

* **Instalación costosa.** Como sistema complejo y externo, no siempre viene integrado dentro del sistema que debemos usar. La implantación de un sistema de base de datos puede llevar consigo un coste elevado, tanto en equipo físico como lógico. Aparte, dado que constituye la parte más crucial y es donde se almacenan los datos, requiere un cuidado especial.
* **Personal especializado.** Siempre que alguien haga uso tanto del mantenimiento como de la toma o inclusión de datos, necesitará que el programador conozca la jerarquía y el lenguaje utilizado. Por tanto, los conocimientos que resultan imprescindibles para una utilización correcta, y sobre todo para una administración adecuada y eficaz, implican la necesidad de personal especializado.
* **Implantación larga y difícil.**Debido a que los sistemas no son estáticos desde su misma creación, la implantación variará, y acabará con una base de datos totalmente distinta a aquella con la que se empezó.
* **Falta de rentabilidad a corto plazo.** En las grandes instalaciones, una base de datos podría no ser rentable ya que es inviable un uso rápido de sus relaciones.

Si bien es cierto que la implementación de una base de datos puede tener sus inconvenientes, a medio o largo plazo son todo ventajas.

No obstante, las ventajas son mucho mayores:

* **Independencia** de datos con respecto a tratamientos, y viceversa: un dato no depende en ningún momento del proceso que lo llama y se puede usar por varios a la vez.
* **Coherencia.** No existe redundancia incontrolada de datos: si los datos no se repiten, no ocuparán el doble de almacenamiento.
* **Disponibilidad.** Los datos no son propiedad de los usuarios, ni de los procesos, los datos son solo parte del modelo.
* **Mayor accesibilidad a los datos y mayor capacidad de respuesta:** a una base de datos bien implementada se accede muy rápido.
* **Mayor valor informativo.** Datos que en un principio parecieran irrelevantes, unidos en una base, posibilitan la generación de datos extra gracias a los volúmenes.

**Niveles de abstracción de una BD**

El nivel más bajo de abstracción es el **nivel físico**. Este nivel describe en detalle cómo se almacenan los datos en los dispositivos de almacenamiento; por ejemplo, mediante señaladores o índices para el acceso aleatorio a los datos. Así, si una base de datos fuese un armario archivador, en el nivel físico se definiría si las carpetas que contienen información relacionada se almacenan en el mismo cajón o si en cada carpeta se especifica dónde existe otra carpeta con información relacionada con la carpeta actual; es decir, en qué otro cajón se encuentra esa carpeta relacionada.  
El siguiente nivel de abstracción es el **conceptual**. Aquí se describen qué datos se almacenan realmente en la base de datos y las relaciones existentes entre ellos; es decir, describe la base de datos completa en términos de su estructura de diseño. El nivel conceptual de abstracción lo usan los administradores de bases de datos, quienes deben decidir qué información se va a guardar en la base de datos.

Consta de las siguientes definiciones:

* Definición de los datos: se describen el tipo de datos y la longitud de campo de todos los elementos direccionables en la base. Los elementos por definir incluyen artículos elementales (atributos), totales de datos y registros conceptuales (entidades).
* Relaciones entre datos: se definen las relaciones entre datos con el objetivo de enlazar tipos de registros relacionados para el procesamiento de archivos múltiples.

El nivel más alto de abstracción es el de visiones. Se refiere a lo que el usuario final puede visualizar del sistema terminado; por tanto, describe al usuario solo una parte de la base de datos. El sistema puede proporcionar muchas visiones para una misma base de datos. En una base de datos, habrá tablas o relaciones que no deban ver determinados usuarios, pero sí otros. Por ejemplo, un director de un centro educativo debe tener acceso a los datos completos de todos los alumnos, pero un alumno no debería ver más información que la suya propia. En este nivel se representarían este tipo de restricciones.

**Tipos de usuarios de un BD**

**Usuarios normales.** Son usuarios no avanzados que interactúan con el sistema mediante un programa de aplicación con una interfaz de formularios, donde se pueden rellenar los campos apropiados del formulario. Estos usuarios también pueden simplemente leer informes generados de la base de datos.  
**Programadores de aplicaciones.** Son profesionales informáticos que escriben los programas de aplicación utilizando herramientas para desarrollar interfaces de usuario, como las herramientas de desarrollo rápido de aplicaciones (DRA), que facilitan crear los formularios e informes sin escribir directamente el programa.  
**Usuarios avanzados (con frecuencia se los llama *usuarios sofisticados*)**. Interactúan con el sistema sin programas escritos, usando el lenguaje de consulta de la base de datos. Los analistas que envían las consultas para explorar los datos en la base de datos entran en esta categoría; ellos mismos usan las herramientas de procesamiento analítico en línea (OLAP, OnLine Analytical Processing), o herramientas de recopilación de datos.  
**Usuarios especializados.** Son los usuarios avanzados (o «sofisticados») que escriben aplicaciones de bases de datos especializadas y adecuadas para el procesamiento de datos tradicional. Entre estas aplicaciones se encuentran los sistemas de diseño asistido por computadora, los sistemas de base de conocimientos y sistemas expertos, los sistemas que almacenan datos de tipos de datos complejos (como gráficos y de audio) y los sistemas de modelado de entorno.  
**Administradores de la base de datos (ABD)**. Son las personas que tienen el control central del sistema de gestión de bases de datos (SGBD). Entre las funciones del ABD se encuentran:

* Definición del esquema de la base de datos.
* Definición de la estructura y el método de acceso.
* Modificación del esquema y la organización física.
* Concesión de autorización para el acceso a los datos.
* Mantenimiento rutinario.

**Sistema de gestión de BD(SGBD)**

Un sistema SGBD (sistema de gestión de bases de datos) es aquel conjunto de programas, lenguajes, procedimientos, etc., que se encargan de la manipulación y suministro de datos almacenados en una base de datos, además de mantener su integridad, confidencialidad y seguridad para cualquier usuario, tenga este o no conocimientos informáticos.

* **Definición de los datos:**el SGBD ha de poder definir todos los objetos de la base de datos partiendo de definiciones en versión fuente para convertirlas en la versión objeto.
* **Manipulación de los datos:** el SGBD responde a las solicitudes del usuario para ejecutar operaciones de supresión, actualización, extracción, entre otras gestiones. El manejo de los datos ha de efectuarse de forma rápida, según las peticiones realizadas por los usuarios, y, gracias a su independencia, debe permitir la modificación del esquema de la base de datos.
* **Seguridad e integridad de los datos:** ante cualquier petición, además de registrar el uso de las bases de datos, también aplicará las medidas de seguridad e integridad de los datos (adopta medidas para garantizar su validez) previamente definidas. Un SGBD debe garantizar su seguridad frente a ataques o, simplemente, impedir el acceso a usuarios no autorizados.
* **Recuperación y restauración de los datos:** la recuperación y restauración de los datos ante un posible fallo es otra de las principales funciones de un SGBD. Se aplicará un plan de recuperación y restauración de los datos que sirva de respaldo.

**La arquitectura a tres niveles ANSI-X3-SPARC**

En 1975, ANSI-SPARC definió la arquitectura con tres niveles para lograr las siguientes metas:

* Que todos los usuarios puedan acceder a los mismos datos, pero con diferentes vistas, adaptadas a sus necesidades y conocimientos.
* Que los usuarios carezcan de la información sobre los detalles de almacenamiento físico de la base de datos. Toda esa información se abstrae para que
* el almacenamiento sea transparente para los usuarios, porque para ellos carece de interés o relevancia cómo se almacenan los datos físicamente, o cuál es el contenedor o formato de almacenamiento.
* Que el administrador de la base de datos pueda cambiar las estructurasde almacenamiento de la base de datos sin variar las vistas de los usuarios.
* Que la estructura interna de la base de datos no se vea alterada por los cambios que se efectúen físicamente.
* Que el administrador de la base de datos pueda modificar la estructura conceptual de la base de datos sin afectar a los usuarios.

**Nivel externo (vistas individuales de los usuarios)**

Una vista de usuario describe una parte de la base de datos que es relevante para un usuario en particular. Excluye los datos irrelevantes, así como aquellos a los que el usuario no está autorizado a acceder.

**Nivel medio (vista conceptual)**

El nivel conceptual es una forma de describir los datos que se almacenan dentro de la base de datos y cómo estos se relacionan entre sí. Este nivel no especifica cómo se almacenan físicamente los datos.  
Algunas características importantes de este nivel son:

* El *DBA* (‘administrador de la base de datos’) trabaja en este nivel.
* Describe la estructura de todos los usuarios.
* Solo el *DBA* puede definir este nivel.
* Aporta una visión global de la base de datos.
* Es independiente del hardware y el software.

**Nivel interno (vista de almacenamiento)**

El nivel interno implica la forma en que la base de datos se configura físicamente en el sistema informático. En él se describe cómo los datos se almacenan en la base de datos y en el hardware del equipo.  
Mediante este esquema en tres niveles, se consigue que el sistema adquiera independencia física y lógica. La independencia física ayuda a modificar el esquema interno sin variar el conceptual, y la lógica capacita a cambiar el esquema conceptual sin necesidad de alterar las aplicaciones ni los esquemas externos (vistas). Por tanto, desliga a los distintos usuarios, y así cada grupo de trabajo puede reunir de un modo diferente los datos para formar entidades o asociaciones diversas, lo que ayuda a cambiar atributos, asociaciones, entidades, propiedades de los atributos, propiedades de las entidades.

**Funcionamiento de un sistema de gestión de base de datos**

* El proceso lanzado por el usuario **llama al SGBD**indicando la porción de la base de datos que se desea tratar. Por ejemplo, la aplicación o proceso pide acceder a los datos de los productos que se desea comprar en una tienda *online*.
* El SGBD **t raduce la llamada a los términos del esquema lógico**de la base de datos. Accede al esquema lógico comprobando los derechos de acceso y la traducción física (normalmente los metadatos se guardan en una zona de memoria global y no en el disco). Se comprueba si la aplicación y el usuario tienen acceso a los datos que se están demandando y se comprueba dónde se encuentran esos datos almacenados físicamente.
* El SGBD **obtiene el esquema físico**, en el que se indica qué fichero o ficheros contienen la información solicitada.
* El SGBD**traduce la llamada a los métodos de acceso del sistema operativo**que permiten acceder realmente a los datos requeridos. Es decir, básicamente, invoca al sistema de ficheros para recuperar los ficheros en los que se encuentren los datos.
* El sistema operativo **accede a los datos**tras traducir las órdenes dadas por el SGBD. Una vez los encuentra, los obtiene.
* Los datos **pasan del disco a una memoria intermedia**o *buffer*. En ese *buffer* se almacenarán los datos según se vayan recibiendo. Se recupera de manera secuencial el flujo de información binaria necesario hasta recuperar todos los datos solicitados.
* Los datos**pasan al área de trabajo del usuario**(ATU) del proceso del usuario. Los pasos 6 y 7 se repiten hasta que se envíe toda la información al proceso de usuario. Al terminar este paso, los datos ya se encuentran en la aplicación, listos para ser preparados y presentados de forma legible para el usuario.
* En el caso de que haya errores en cualquier momento del proceso, el **SGBD devuelve indicadores**en los que manifiesta que existen errores o advertencias que se deben tener en cuenta. Esto se remite al área de comunicaciones del proceso de usuario. Si, por el contrario, las indicaciones son satisfactorias, los datos de la ATU serán utilizables por el proceso de usuario.

**Lenguajes de las bases de datos**

Los lenguajes de las bases de datos se definen de dos maneras:  
**1. Lenguajes de definición de datos**(DDL). Son los que permiten describir y nombrar entidades, atributos y restricciones.  
**2. Lenguajes de manipulación de datos**(DML). Son los que permiten manipular los datos contenidos en la base de datos (inserción, modificaciones, extracción o borrado). Estos lenguajes se dividen a su vez en dos grupos:

* **A. Procedimentales.**El usuario puede especificar qué datos necesita y la forma exacta de extraerlos. Extrae un registro, lo procesa y, dependiendo de los resultados, extrae otro registro para procesarlo de forma similar. Suelen estar integrados en un lenguaje de alto nivel para facilitar la iteración y gestión de la lógica de navegación.
* **B. No procedimentales.** El usuario puede indicar qué datos necesita, pero no cómo extraerlos (a este tipo de lenguajes se les denomina *lenguajes declarativos*).

El SQL es ilustrativo como ejemplo, pues combina ambos lenguajes en sus sentencias, y estas se pueden agrupar:  
**A)** Las sentencias DDL se utilizan para crear y modificar la estructura de las tablas, así como otros objetos de la base de datos:

* + ***Create:*** crea objetos en la base de datos.
  + ***Alter:*** modifica la estructura de la base de datos.
  + ***Drop:*** borra objetos de la base de datos.
  + ***Truncate:*** elimina todos los registros de la tabla, incluyendotodos los espacios asignados a los registros.

**B)** Las sentencias de lenguaje de manipulación de datos (DML) son utilizadas para gestionar datos dentro de los *schemas*. Algunos ejemplos:

* + ***Select:*** para obtener datos de una base de datos.
  + ***Insert:*** para insertar datos a una tabla.
  + ***Update:*** para modificar datos existentes dentro de una tabla
  + ***Delete:*** para eliminar todos los registros de la tabla; no borra los espacios asignados a los registros.

**Modelo de datos**

Vamos a trabjar con el modelo de datos es el tipo de datos esto debería poderse tratar con el lenguaje que estemos creando nuestro programa

**Concepto modelzacion de datos**

Todo sofare debe ir dirigido a un espacio en concreto y el programador debe decir que hace parte del entorno los posibles problemas y los que no lo van hacer

Vamos a explicar que es el universo de discurso con un ejemplo

Digamos que vamos a crear una tienda asi que creamos un párrafo que explica la situación y una lista con todos los elementos que precisemos necesitar.

**Modelos y esquemas**

Esto hace parte de la organización que la daremos al proyecto debemos tener en cuenta la distribución de los esquemas tenerlos bien claros y no mezclar esquemas. Esto hace que sea un conjunto de reglas entendidas tanto por el receptor y el emisor.

Debemos destacar la diferencia entre modelo de daos y es quema.

El es quema representaría donde guardar los datos

Y el modelo de datos nos mostraría las distintas partes y relaciones entre los datos

**Clasificaion de modelos de datos**

Según el nivel de abstración que se considere en referencia a la arquitectura **ANSI/X3/SPARC**

Los clasificamos en:

1 **externos** -> orientado a cada usuario en particular define lo que puede y no puede hacer cada usuario

2 **globales ->** orienttado a un grupo de usario nos da una vista global del proyecto

3 **internos** -> son la representación del modelo a mas bajo nivel nos dice en que estructura de datos se va a utilizar

**Modelos lógicos(globales y externos)**

**Se divienden en los siguientes puntos**

* **Conceptuales o semánticos**  enfocados a descubir el mundo real con independencia de la tecnología principales características
  + representan sus elementos con nodos conectados
  + faciles de comprender
  + utilizados en informática e ingeniería
* **orientado a objetos** diagramas de clase UML
  + no suelen estar representados en SGBD
  + poseen un mayor nivel de abstración
  + cuentan con mayor capacidad semántica
  + orientados a diseño de alto nivel
* **convencionales** enfocados a la implementación en SGBD
  + implementados en SGBD comerciales
  + dependen del SGBD
  + son mas próximos al ordenador
  + interfaz informatico sistema

**componentes de un modelo de datos**

**formalizacion de modelos de datos**

un modelo de datos (md) de define como par **MD=>S,D>**

* **S O ESTATICA**  condjunto de reglas de generación que permiten representar la componente estatica. Permite formalizar la especificaión de estructura admitida
  + - **Elementos permitidos (SE)**  objeto y vínculos enctre objetos ejemplo si hay dos personas casadas en la base de datos. Las dos personas seria el objeto de clase persona mientras el estar casado supondría un vinculo
    - **Elementos no permitidos o invalidados** (SR) permite definir las estrucutras e invalidar determinados estados en la base de datos
* **D O DINAMICA**  es ell conjunto de operaciones autorizadas sobre la componente estatica operaciones que permiten representar la componente dinámica ejemplo crea un nuevo usuario

**Estatica de los moldes de datos**

1. **Tipos de estructuras** 
   1. Los elementos mas comunes en los modelos de datos son los objetos , vínculos entre objetos, propiedades o caracteriticas de los objetos y dominios conjuntos de valores homogéneos admisibles sobre los que se definen el as propiedades se pueden representar en esquemas diagramas o arboles
   2. **Mecanismos de abstracion**  es el esquema de datos que resulta de aplicar un proceso de abstracion sobre el mundo real de los datos
      1. Clasificación o generalización se generan características comunes de una serie de instancias para crear una categoría superior o generalizada.
      2. Agregación agregar un nuevo elemento
         1. Agregación de partes individuales del (md) ejemplo un plus en el salario de un empleado
         2. Agregación de propiedades para obtener una propiedad compuesta esto no todos los modelos lo admiten
         3. agrupacion de clases para obtener una clase compuesta veihiculos anfibios un super clase para contar tnto consumo en tierra como en agua
      3. Jerarquía de generalización suponen un refinamiento de la abstracion de generalización
      4. **Asociación**  abstracion que se utiliza para vincular dos o mas clases
      5. **Rectriciones**  prohibición de una base de datos para evitar comportamientos indeseados
         1. **Restricciones estructurales**
         2. **Restricciones de integridad semántica (ris) o explicitas**  son restricciones de rango

**Y se categorizn como**

* **Propias al md (SEMANTICA INTEGRADA)** su definición corresponde al diseñador pero su gestión es responsabilida del modelo de datos
* **Ajenas el md (SEMANTICA DISPERSA )** vienen definidas por el diseñador y sueles er especificas en cada base de datos

**Depende si se tiene que definir aciones e harán en**

* **Acciones generales** se lleva acabo por programa determinados se dividen
  + **Procedimiento de almacenado**
  + **Disparadores**  se formula una declaración que si se cumple una condición se dispara la acción
* **Acciones especificas**  la acción esta implícita en la propia restricion se dividen en
  + **Condición general verificación y aserción**
  + **Verificaion**  la expresión lógica mediante la cual se formula la expresión
  + **Aserción tiene existencia por si misma**
  + **Condición especifica**  se refiere a las disversas opciones que que facilitan los distinto md ejmplo primary key freing key not null

**Parte de una resticion**

se realiza la operación.

La condición ha decumplirse

La acción debe llevarse acabo tras la evaluación de la condición

**Metabase de datos**

Cuando se aplica la componente estatica a un modelo de datos (**MBD)**

La metabase de datos esta compuesta por las estructuras del esquema (**EMBD las tablas que tenemos )**  y las restricciones de integridad definidas(**RMDB** las reglas definidas para cada dato )

**Dinámica**

Es una BD que se refiere a los movimientos o los cambios en esta, estas son eglas con palabras clave reservadas llamadas operaciones. La consecuencia de las ordenes se le conoce como transacciones

**Operadores**  manejan el modelo de datos se suelen expresar como

LOCALIZACION <CONDICION> ACCION <OBJETO>

**Localización o enfoque:**  consiste en localizar una instancia de un objeto indicando un camino

**Condición**  representa un condición lógica que deben cumplir los objetos que desean cumplir o llega a esa ubicaion

**Transacciones**  modificación de los datos estas deben cmpletarse correctaente

**Punto de sincronismo**  aspecto importante en cuanto a integridad de una base de datos es el principio de atomicidad de una transsacion

* Cada transacioon se debe mostrar como un bloque o unidad y debe garantizarse que el final sea consistente
* Cuando la transacion llegue a estado commit se dará por terminada
* Sino ocurria un abort la transacion nunca dee quedar en un estado intermedio o final

**Modelo entida-relacion o entidad-interrelación**

Peter Chen definió en 1976/1977 un modelo de datos conocido como *entidad-interrelación* (‘*entity/relationship model’*, o ERM)

Eso sí, el ERM representa un buen ejemplo en el apartado de estructuras y restricciones (estática), pero no en el componente dinámico. Algunos autores han intentado dotarlo de dinámica —lenguajes gráficos de recorrido del grafo que representa un diagrama entidad-relación (ERD)

**Estructura**

En este modelo se hace distinción entre los siguientes elementos, que configuran la estática del modelo en dos categorías:

1. Dominio conceptual superior:

* A. Entidad (*entity*).
* B. Interrelación o relación (*relationship*).

2. Dominio conceptual inferior:

* A. Dominio (*domain*).
* B. Atributo (*atribute*).

#### Diagramas entidad-relación (ERD)

Chen pensó en una forma gráfica para poder modelar los conceptos y las reglas necesarias para cada modelo. Esta notación paso a llamarse *diagrama entidad-relación*(‘*entity relationship diagram’* o ERD), donde cada parte del grafo obedece a conceptos del modelo que se explicarán en los siguientes párrafos.

**Entidad (‘entity’)** puede considerarse entidad cualquier objeto real o abstracto dependiendo del nivel de abstración podemos decir que hay dos tipos de entidad

* Cada entidad es distinta de otra y posee sus propios atributos
* Cuando se abstrae el concepto grupo de entidades se le llama entida de tipo entitytype

Las entidades en ERD se representan con un sustantivo en singular y una caja o rectángulo:



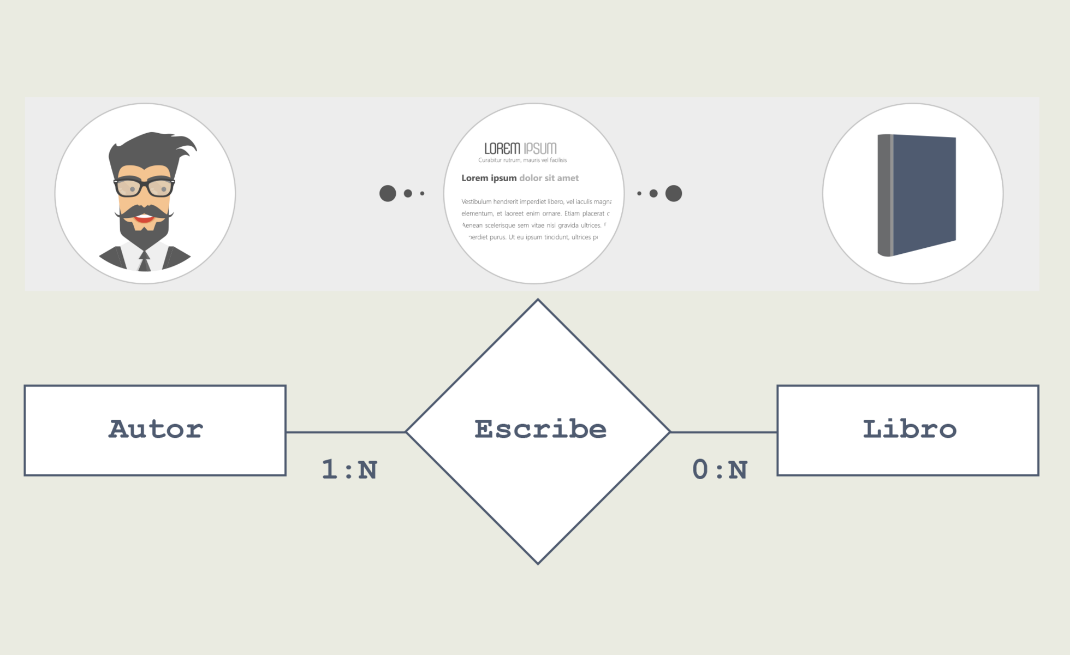
Dentro de las entidades es posible diferenciar dos categorías, dependiendo de su existencia:

* **Regular o fuerte *(‘regular’)***. Las entidades fuertes existen por sí mismas y no dependen de otras para existir, como en los ejemplos de las asignaturas o de las personas.
* **Débil *(‘weak’)***. Las entidades débiles son aquellas que dependen de una entidad anterior para existir; sin ella carecerían de sentido. Por ejemplo, el número de la edición de un libro aporta información (cambios, quién lo edita…); pero, si no existe dicho libro, son entidades sin sentido. Las entidades débiles se expresan con una caja doble:

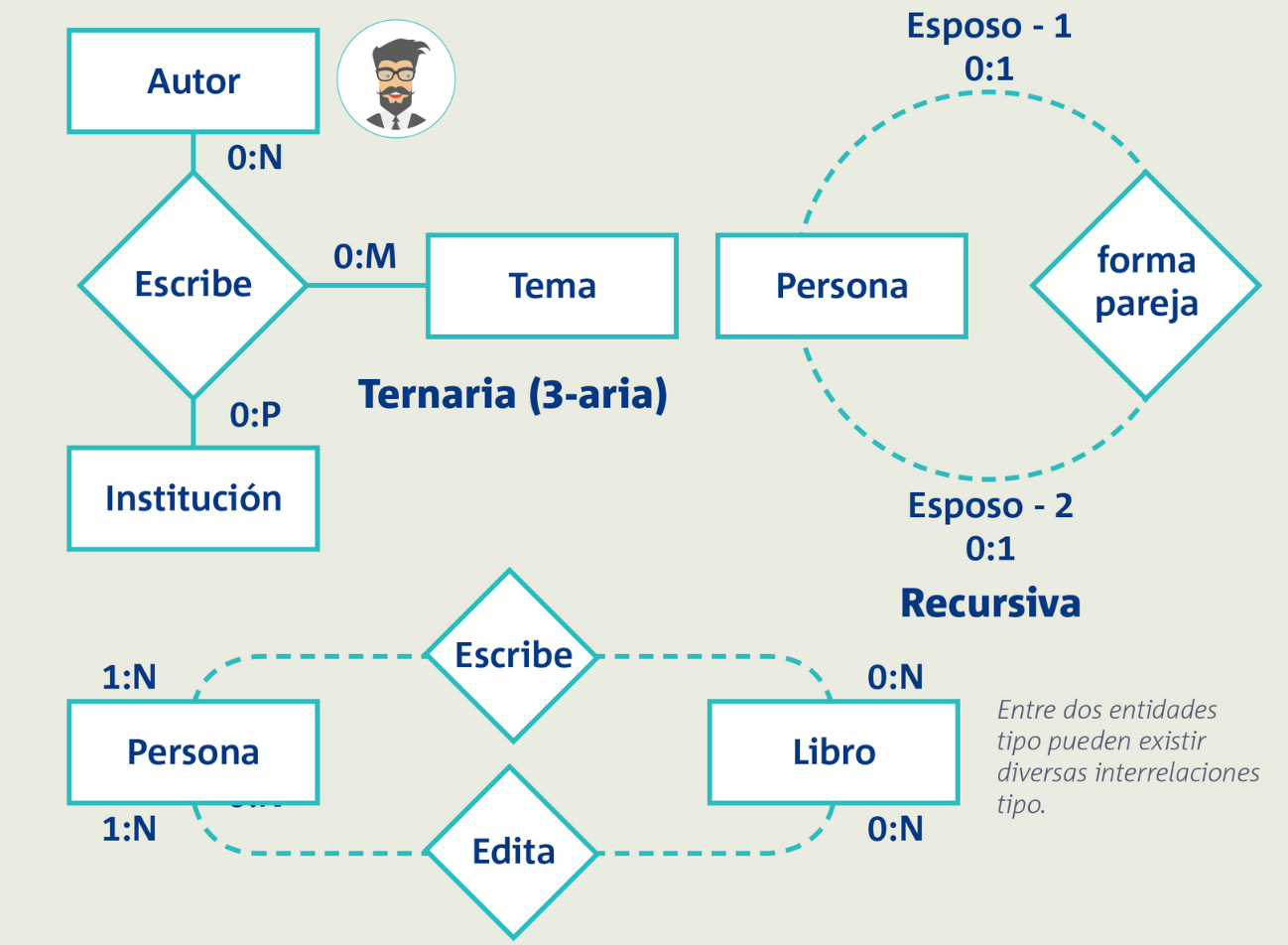


**Interrelación (‘*relationship’*)**

Una interrelación, como su propio nombre indica, se refiriere a la relación entre dos entidades. Al igual que ocurrió con entidades y entidades tipo, en interrelaciones existen la interrelación y la interrelación tipo. Y, de la misma manera, la interrelación tipo representa la abstracción de la interrelación. Matemáticamente, se define como:

**{<e1, e2…, e** ***n*** **>},**  
donde e*i* es un ejemplar del tipo de entidad E*i*, y *n* el grado del tipo de interrelación; es decir, *n* refleja el número de tipos de entidades participantes que hay asociadas en este tipo de interrelación. interrelación. 

* 1. **Cardinalidad**  
     La cardinalidad define el número de casos que pueden ocurrir en una interrelación: 1:1, 1:N, N:M, 0:M, 0:1, etc. Por ejemplo, un autor puede haber escrito de 0 a N libros, y un libro puede ser escrito por un único autor (1) o por varios (N).
  2. **B) Rol**  
     Un rol es una función que desempeña una entidad en una interrelación tipo. No son necesarios, pero sirven como aclaración en las interrelaciones.  
     Un caso típico en el que se requiere precisar el rol de cada tipo de entidad participante se da cuando existe una interrelación reflexiva; es decir, cuando hay un tipo de entidad asociado consigo mismo.  
     Un ejemplo de ello es cuando hay una interrelación entre objetos del mismo tipo, como puede ser una lista de personas. Allí una persona puede ser cliente de otra persona que es, a su vez, el vendedor del servicio. Si solo se ha definido la persona, cada unión se llamará *vendedor*o *cliente* . En estos casos, conviene también especificar el grado, ya que, por ejemplo, vendedor solo puede ser uno, y la relación 1:1; pero clientes hay muchos, luego la relación sería 1:N.
  3. **Grado**  
     El grado hace referencia al número de entidades que unen la interrelación. Las más comunes son las binarias (grado-2), pero también pueden ser reflexivas (grado 1), ternarias (grado 3) o *n*-arias (grado *n*).  
     A mayor grado, más complejo se volverá el modelo para generar bien las interrelaciones.



Así, en esta figura se muestran tres tipos de relaciones. La primera es *forma pareja*, una relación reflexiva (también llamada *recursiva*), porque relaciona instancias de una entidad con instancias de la misma entidad. Segundo, *escribe* y *edita*, dos relaciones binarias. Por último, una ternaria (*escribe*), en la que, además de relacionar *autor* con *libro*, se quiere almacenar información sobre la *institución* en la que dicha relación se produce.  
La lectura de este gráfico modela una realidad en la que:

* Persona escribe de 0 a N libros. Libro es escrito por 1 a N personas.
* Persona edita de 0 a N libros. Libro es editado por 1 a N personas.
* Persona forma pareja con 0 o 1 persona.
* Autor escribe 0 a M temas en 0 a P instituciones. Institución escribe de 0 a M temas con 0 a N autores.

#### Dominios y valores

Cada vez que se da a una entidad tipo o relación tipo un significado específico, se dice que esa entidad posee ese **valor**. Por otro lado, un **dominio** se define como «un conjunto de valores homogéneos con un nombre que lo identifica».  
Una cierta característica o propiedad de un objeto toma valores que pertenecen a un determinado dominio. Por ejemplo, la propiedad *número de páginas en un libro* puede tomar valores pertenecientes al dominio de los números naturales. Su notación matemática es:

**D = {v*i* : p(v*i*)},**

donde D es el dominio, v*i* un valor y p el predicado asociado a dicho dominio. Un dominio puede definirse de una de las siguientes maneras:

* Por **intención**, especificando el tipo de datos. Por ejemplo, a la hora de almacenar un número de teléfono de España, se especifica que el dominio del campo *teléfono* en la base de datos solo puede contener nueve números enteros (o un número entero de nueve cifras).
* Por **extensión**, enumerando los valores que pertenecen al dominio. Por ejemplo, en el campo *día*, enumerando los días de la semana: *lunes, martes, miércoles…*

**Propiedades o atributos**

Este apartado se refiere a cada una de las propiedades o características que describen un tipo de entidad o un tipo de interrelación.  
Los atributos **toman valores de un dominio**(atributos simples) **o de varios dominios** (atributos compuestos). Por ejemplo, una contraseña en la que solo se permitan números, como pasa con el pin de una tarjeta de crédito, o una contraseña que permita combinar números y letras. Matemáticamente, un atributo consiste en una función que aplica un tipo de entidad o de interrelación sobre todos los posibles subconjuntos de los valores de un dominio (o de un conjunto de dominios):  
A : E → S(D), o A : E → S(D1) × S(D2)… × S(D*n*);  
A : I → S(D), o A : I → S(D1) × S(D2)… × S(D*n*),

**Propiedades identificadoras**

Dado que una entidad siempre puede distinguirse de las demás, entre todos los atributos de un tipo de entidad existen siempre uno o varios atributos (simples y/o compuestos) que identifican unívocamente cada una de las instancias de ese tipo de entidad.  
Al conjunto de atributos que permiten identificar o distinguir una entidad de otra se le denomina ***identificador primario (IP)***. Por ejemplo, es muy común utilizar el DNI de una persona como identificador primario.  
Una entidad puede tener otros conjuntos que permiten, adicionalmente, identificar a las entidades; a estos se les denomina ***identificadores alternativos (IA)***. Por ejemplo, usar un conjunto creado por *fecha de nacimiento, hora de nacimiento*y *grupo sanguíneo* como identificador alternativo.

**Restricciones de integridad**

1. **Restricciones inherentes** Solo es posible establecer interrelaciones entre entidades; por tanto, no se admiten entidades entre interrelaciones ni interrelaciones entre interrelaciones
2. **A) Dependencia en existencia**  
   Las entidades débiles en existencia son aquellas que no pueden existir sin una interrelación, ya que sin esta no tendrían sentido. Estas entidades quedan definidas por una **interrelación especial**: la que induce la dependencia de esta entidad con otra de orden superior (la de orden superior puede ser una entidad fuerte o débil). Esto quiere decir que toda entidad débil posee una **dependencia en existencia** de la entidad de orden superior, por lo que entre ellas se define una jerarquía de dos niveles.

**B) Dependencia en identificación**  
Existen entidades que se denominan *débiles en identificación*, y son aquellas que no se pueden identificar solas, en sí mismas. En la identificación se obligará a que tome parte la entidad de la que depende y, además, se agregarán a la identificación atributos de la entidad débil. Por ejemplo, en una empresa no es posible identificar un pedido solo por los productos y las cantidades; es preciso asociar a él algún identificador sobre el almacén que realiza el pedido. Los almacenes se identifican mediante *IdAlm*.

* Se quiere mantener una lista de pedidos numerados en cada almacén (en un intervalo entre 1 y 999999).
* Dicho intervalo es el mismo en cada almacén, así que podría existir el pedido 20 340 en el almacén IdAlm = 1 e IdAlm = 2. De ahí que la identificación de un pedido sea la composición *<IdAlm&IdPed>*, por lo que las instancias quedan identificadas mediante *<1,20340>*y *<2,20340>*.
* Como se aprecia en la imagen, la dependencia en identificación se caracteriza por un rombo con doble trazo y un arco orientado a la entidad débil.

Como ya se habrá deducido, **una dependencia en identificación implica también dependencia en existencia.**

**C) Restricciones sobre valores**  
Se establecen mediante la definición del dominio o tipo de valor. Por ejemplo, si el dominio del atributo *número de teléfono* es de tipo entero, ningún número de teléfono de la base de datos podrá contener letras o caracteres que no sean números enteros.  
**D) Restricciones sobre atributos**  
Las restricciones sobre atributos permiten diferenciar atributos identificadores primarios (IP) y alternativos (IA). El modelo no permite especificar otro tipo de restricciones sobre el resto de los atributos. Este tipo de restricciones son únicas en cada entidad. Es decir, no puede haber más de una restricción de identificador primario, también llamado *clave primaria*. Por ejemplo, si el IP de una entidad lo conforma el DNI, no puede haber otro atributo al que se le asigne la restricción de IP.  
**E) Cardinalidades**  
Se pueden establecer restricciones de cardinalidad de interrelaciones y sobre roles. Las primeras son redundantes si se especifican las cardinalidades de roles. Por ejemplo, es posible forzar y controlar que un empleado no disponga a su cargo a más de tres empleados. Y, si alguien introduce una instancia en la entidad que rompe esta restricción, será notificado por el sistema y dicha instancia no se guardará en la base de datos.

### **Modelo relacional de Codd**

Gracias a las reglas postuladas en 1970 y al modelo de Edgar Frank Codd, IBM consolidó y materializó muchos conceptos abstractos, y así creó un paradigma práctico bajo el cual podían almacenarse los datos de manera consistente. Por eso, no tardó en convertirse en una de las mejores formas de modelar e imaginar una base de datos. En este modelo, tanto las entidades como las relaciones se imaginan como tablas en las que se registra la información por filas (también llamadas *tuplas*). En estas tablas, las columnas (también llamadas *campos*) representan los atributos de la relación o de la entidad. A continuación, se definirá formalmente cada uno de estos conceptos:

**Estructuras**

**A) Relación**  
Es la estructura básica del modelo relacional. Con una relación es posible representar tanto instancias de una entidad del universo real como interrelaciones entre entidades de distinto tipo. Además, recoge interrelaciones de cardinalidad múltiple; es decir, permite almacenar todas las instancias de tipo «muchos a muchos». Su representación informal es una tabla, y en el caso particular de esas instancias de cardinalidad múltiple, existirá una fila (tupla) por cada instancia que almacenar.

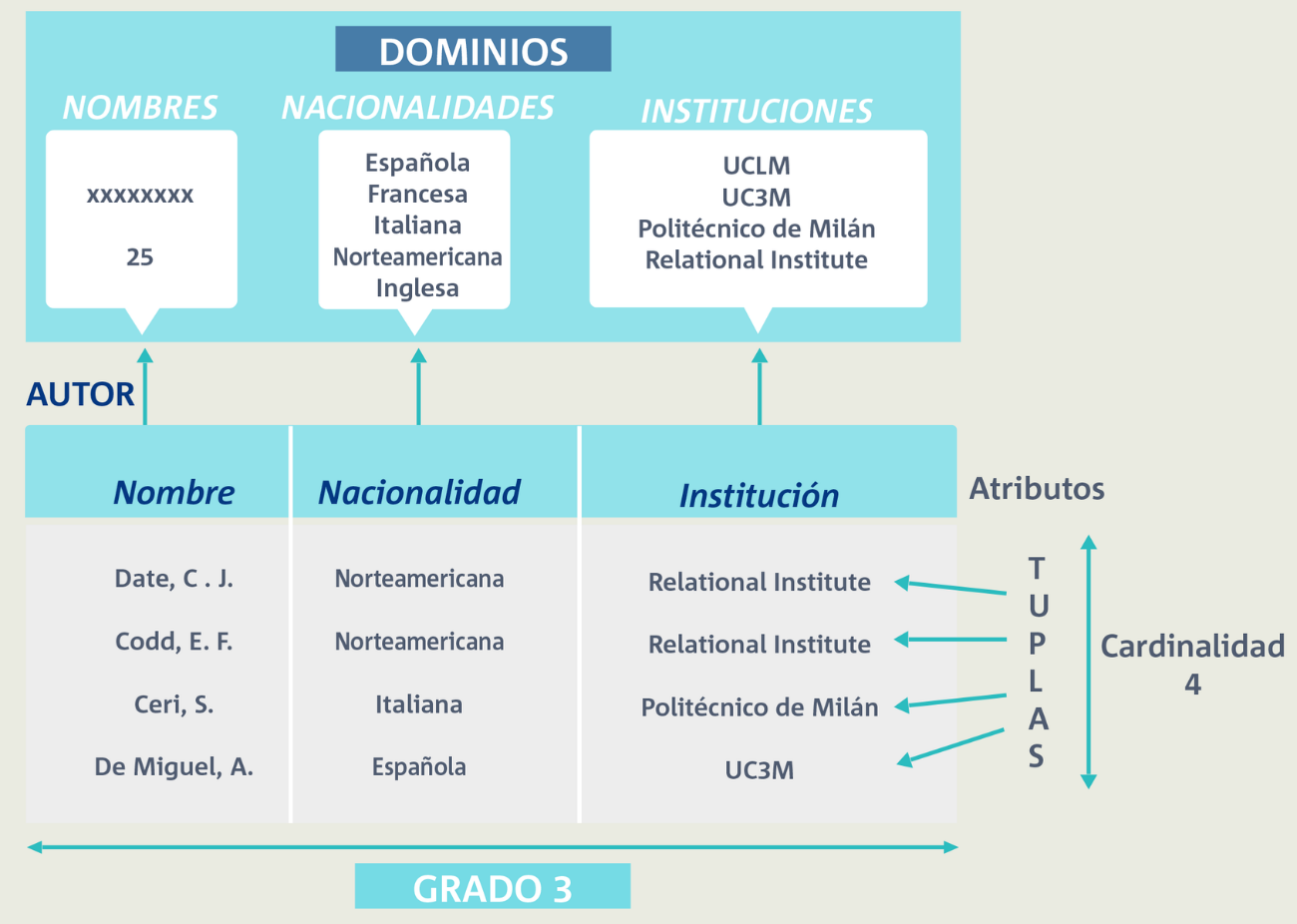
**B) Dominio**  
Como se vio en el apartado 4.1.4., «Dominios y valores», un dominio es el conjunto válido de valores de referencia para definir propiedades o atributos. Se trata de un conjunto nominado y homogéneo de valores. Por ejemplo, los caracteres del alfabeto como dominio para un campo como *nombre propio*.  
Un dominio compuesto se puede definir como una combinación de dominios simples a la que se pueden aplicar ciertas restricciones.  
**Ejemplo**: el dominio compuesto denominado *fecha* se construye por agregación de los dominios simples *día, mes*y *año*. En el dominio compuesto se deberían añadir restricciones, a fin de que no aparezcan valores inválidos, como *29/2/2003* o *31/4/2004*. Es decir, 29 es válido como día del mes, pero solo en un febrero cada cuatro años. Del mismo modo, abril es un mes de 30 días, por lo que, aunque 31 sea un valor correcto para *día del mes*, no lo es para todos los meses. El dominio agregado *fecha*permite controlar este tipo de casuísticas.

**C) Atributo**  
Representa las propiedades de la relación o entidad. Un atributo, necesariamente, ha de definirse sobre un dominio. Su representación informal es una columna. Por ejemplo, en una hipotética relación llamada *préstamo bibliotecario* —que relaciona a un usuario con un libro—, un atributo (*columna*) de la relación podría ser *número de días* del préstamo. Por tanto, un atributo (A) es la interpretación de un determinado dominio en una relación; es decir, el «papel» que juega el dominio en dicha relación.

**D = Dom(A) ⇒ D es el dominio de A**

Continuando con el ejemplo anterior, *número de días* del préstamo sería un valor tipo numérico —concretamente, del dominio de los números naturales, ya que no puede contener ni números negativos ni decimales—. Es decir, los días del préstamo pueden ser 0, 1, 2, etc.; pero no −1 ni 0,5 días.

* Un atributo está siempre asociado a una relación, mientras que un dominio tiene existencia propia, con independencia de las relaciones que existan en el modelo.
* Un atributo representa una propiedad de una relación.
* Un atributo toma valores de un dominio.
* Varios atributos distintos (de la misma o de diferentes relaciones) pueden tomar sus valores del mismo dominio.



Ejemplo completo de una relación.

**E) Grado**  
Se define como el número de dominios: D*i*.  
**F) Cardinalidad**  
Se define como el número de tuplas de la relación.  
**G) Diferencias entre relaciones y tablas**

* Tabla:
  + 1. Poseen un orden definido para las filas.
  + 2. Hay un orden definido para los valores de los atributos.
  + 3. Pueden repetirse filas.
* Relación:
  + 1. Ni las tuplas ni los valores de los componentes de una tupla se referencian por su posición relativa (no hay un orden predefinido para el conjunto).
  + 2. No pueden repetirse tuplas, pues un conjunto no tiene elementos repetidos.

#### Contexto matemático del modelo relacional

Con objeto de conseguir una granularidad más fina y mejorar la independencia de datos, Codd propone relajar la hipótesis de correspondencia posicional de valores de atributos en una tupla respecto a sus dominios. Es decir, propone que la posición (número de columna) que ocupa un atributo dentro de una tupla (fila de la tabla) no sea importante (a eso se refiere con *relajar*); sino que lo importante sea cómo se denomine ese atributo, ya que es por ese nombre (y no por la posición que ocupa) por el que se buscará al atributo a la hora de recuperar datos guardados o guardar nuevos datos en dicha tupla.

El término *granularidad* se refiere al nivel de detalle con que se almacena en la base de datos. Por ejemplo, en una BD que almacene las visitas a una web, puede guardarse el número de visitas cada día, pero también afinarse (reducirse) la granularidad y almacenar datos para cada hora, e incluso para cada minuto. También es posible ampliarla guardando las visitas cada semana o cada mes. Aumentar la cantidad de datos guardados mejora la capacidad de análisis, pero también aumenta el tamaño de la BD.

Se redefine una relación como:

* Un nombre de relación.
* Un conjunto de atributos A = ({A*i*}), donde el número de atributos es el grado de la relación.
* Un conjunto de dominios D = ({D*i*}).
* Una aplicación que asocia a cada atributo de A un dominio D. **Dom : A  → D.**

Con esta definición no hay una relación posicional de valores/dominios, sino una correspondencia atributo/valor dentro de cada tupla (por ejemplo, el valor que contenga el atributo *DNI* podrá ser recuperado correctamente, con independencia de la posición en la que se encuentre). Gracias a ello, se ha conseguido granularidad a nivel de atributo elemental (porque cada elemento es independiente de su posición y puede ser recuperado él solo, y no junto con otros atributos). Se recuperan los valores en una tupla por la denominación del atributo, en vez de por su posición. Los cambios en el esquema de la relación que impliquen adicionar nuevos atributos no afectarán a programas de aplicación que accedían a un subconjunto de valores de una tupla.

**4.3.3. Restricciones**

**A) Concepto intuitivo de *identificador***  
Al manejar cualquier estructura de datos se hace necesario distinguir elementos en conjuntos de datos. Un identificador, en general, es un subconjunto de propiedades que permite distinguir o identificar elementos dentro de dicho conjunto.  
Normalmente, pueden existir distintos identificadores en una estructura de datos. Es simplemente la propiedad o conjunto de propiedades que identifican unívocamente algo en un conjunto.  
**B) Superclave**  
En el modelo relacional, como una relación es un conjunto, no puede haber elementos duplicados. Por tanto, está implícita la existencia de, al menos, un identificador para las tuplas de la relación. Una superclave cumple esta condición de unicidad. Se podría decir que **una superclave es el conjunto de identificadores que garantiza la unicidad** y, por tanto, identifica un elemento del conjunto de forma única; es decir, el conjunto de identificadores garantiza que un elemento es único en el conjunto de elementos.  
Una superclave puede no ser la única ni la mejor. Y, en un conjunto, pueden existir varias superclaves diferentes agrupando otros identificadores. Por ejemplo, una superclave podría ser el nombre y apellidos junto con la fecha de nacimiento y DNI.  
**C) Clave candidata**  
Una clave candidata es aquella superclave que cumple las condiciones de unicidad y minimalidad.  
De la definición de relación se deriva que **siempre existe, al menos, una clave candidata**: al ser una relación un conjunto y no existir dos tuplas iguales, el conjunto de todos los atributos siempre tiene que identificar unívocamente a cada tupla. Es decir, para obtener una clave candidata, primero hay que identificar las superclaves y, segundo, eliminar de ellas los identificadores que, si se sacaran del conjunto, no impedirían identificar unívocamente la entidad. En el ejemplo recién usado —el de superclave compuesta por nombre y apellidos, fecha de nacimiento y DNI—, se infiere que se cumple la condición de unicidad: con esos tres atributos se identifica unívocamente cada tupla; sin embargo, no se cumple la minimalidad, porque tan solo con *DNI* se podría identificar

unívocamente una tupla.  
El concepto de *clave candidata* es **muy relevante** en el modelo relacional y representa el mecanismo para referenciar las tuplas de una relación.  
Como ejemplo, imagínese que al diseñar la base de datos de una librería se intuye que una superclave podría ser el conjunto formado por los atributos *Nombre de usuario* y *Número de socio*. Queda claro que este conjunto identifica unívocamente a un usuario. La pregunta es la siguiente: ¿si se saca *Número de socio* del conjunto, se seguiría identificando unívocamente a los usuarios? La respuesta es que no, porque podría ocurrir que dos usuarios se llamasen igual. Entonces, habría que formularse una nueva pregunta: ¿si se saca *Nombre de usuario* del conjunto, se seguiría identificando unívocamente a los usuarios? La respuesta es sí; por tanto, *Número de socio* podría convertirse en una clave candidata (por ser mínima).  
**D) Clave primaria**  
Una relación puede tener más de una clave candidata. La clave primaria (***‘primary key’***) es la clave candidata que el usuario escoge para identificar las tuplas de la relación (siempre existe clave primaria). En el caso recién expuesto, *número de socio* era una clave candidata; pero también podría serlo el atributo *DNI*, ya que este también identifica unívocamente a un socio. Cualquiera de las dos podría convertirse, en principio, en clave primaria.  
**E) Claves alternativas**  
Las claves alternativas (***‘alternative keys’***) son las claves candidatas que no han sido escogidas como clave primaria. En el ejemplo anterior, si se eligiera *número de socio* como clave primaria, entonces *DNI* se convertiría en una clave alternativa.  
**F) Claves ajenas**  
Las claves ajenas (***‘foreing keys’***) son el mecanismo de referencia del modelo relacional que permite representar interrelaciones 1:1 y 1:m. Lo hace a través de referencias a otras entidades o relaciones: incorpora un atributo que hace referencia a la clave primaria de otra entidad o relación.  
También es posible representar interrelaciones múltiples (m:n). Para ello es preciso definir una nueva relación cuya clave primaria se componga de las claves primarias de las relaciones referenciadas. De este modo, dichas claves actuarán a la vez como claves ajenas de esas relaciones referenciadas.  
Por **ejemplo**, en la librería, podría haber la relación *préstamo*, en la cual se almacenara que el *libro L* ha sido prestado a la *persona P* mediante una tabla que relaciona el *número de serie* del *libro L* con el *número de socio* de *la persona P*. Como en esa relación no se almacena el nombre, no se conocen más datos de P que su número de socio; pero se necesita saber cómo se llama o dónde vive para notificarle la devolución. Estos datos se almacenan en otra tabla, a la cual es posible acceder si en la relación *préstamo* se almacena el *DNI* como clave ajena, la cual hace referencia a la clave primaria (*DNI*) de la entidad *socio* , en la que es posible almacenar el nombre completo, dirección, teléfono de contacto, etc. Este no es el único mecanismo para hacer estas referencias, pero sirve de ejemplo para aclarar estos conceptos.

#### Reglas de integridad del modelo relacional

**A) Valor *null*** **en el modelo relacional**  
Se define como *null* (‘nulo’) el valor que puede tomar un atributo en una tupla cuando se desconoce dicho valor. Por ejemplo, el valor *null* podría expresarse como «No se sabe el domicilio de una persona», o «Se desconoce su fecha de nacimiento», etc. Se diferencia el valor *null* de cualquier otro valor (blanco, --, etc.) que implique una asignación a un atributo; porque *null* significa que dicho atributo no tiene asociado ningún valor o que su valor es desconocido. *Null* puede asignarse a atributos en cualquier dominio; ahora bien, no es equivalente el valor *null* de un atributo tipo con el de otro. Por ejemplo, el valor *null* de *fecha*no es equivalente con el de *domicilio*; es decir, no se pueden comparar nulos de atributos distintos. En cambio, sí puede evaluarse si el valor de un atributo es o no *null*.  
Véase el siguiente caso:  
Alumnos ( **DNI**: D\_DNI, **nombre**: *string*, **domicilio**: *string*, **fecha**\_nac: *date*) PK(DNI)

Ejemplo de tuplas:  
<01234567S, “Antonio Sánchez Sánchez”, C/Trabajo 22, null>  
<87654321S, “Antonio Márquez Márquez”, null, #12/12/1982#>  
<00000123S, “Antonio Márquez Sánchez”, null, null>

**B) Integridad de la entidad**  
La regla de integridad de la entidad es el mecanismo que gobierna el comportamiento de la clave primaria de una relación. Y lo hace de modo que, en el modelo relacional, la representación de entidades garantice la identificación o distinción de tales entidades en el universo de modelado. La regla declara lo siguiente: «**Ningún atributo que forme parte de la clave primaria de una relación puede tomar un valor nulo**»; es decir, no puede adoptar un valor desconocido o inexistente.  
Véase el siguiente ejemplo:

* **Si PK es un atributo elemental**—por ejemplo, el DNI del alumno—, este no puede tomar el valor *null*. Si esa propiedad fuera desconocida, significaría que, en el mundo modelado, no se conoce quién es el alumno. Y por tanto tampoco podría distinguirse de otro alumno. Esta regla se aplica con independencia de que solo haya una tupla con ese valor.
* **Si PK está formada por un descriptor compuesto**, habría que indagar en ese compuesto para ver si es realmente una clave primaria. Por ejemplo, con el fin de identificar pedidos en una tienda realizados por un cliente, se ha elegido el siguiente subconjunto de atributos: (*tienda, DNI: cliente, fecha\_pedido*). Si se estimara que en algún caso —es decir, en alguna tupla— bastara con *tienda*y *DNI: cliente*para identificar dicha tupla, entonces podría desecharse *fecha\_pedido*como identificador de las tuplas del pedido. Esto contradiría el que *(tienda, DNI: cliente, fecha\_pedido)*fuera una clave primaria, pues no cumpliría la condición de minimalidad. Por tanto, *(tienda, DNI: cliente, fecha\_pedido)*sería en realidad una superclave.

La condición de minimalidad es aquella que garantiza que una superclave puede ser clave candidata por estar formada por el número mínimo de atributos que garantizan la identificación unívoca de una tupla.

**C) Integridad referencial**  
La regla de integridad referencial gobierna las situaciones de representación de interrelaciones en el universo de discurso. Y lo hace de modo que se representan vínculos existentes o inexistentes, pero se evitan referencias no permitidas.

* **Si FK referencia una tupla**—por ejemplo, la editorial del libro—, el valor que adopte ha de corresponder necesariamente a una tupla de editorial existente en la relación de editoriales (siempre que ese valor no sea nulo, claro). No se admite un valor que no esté debidamente referenciado; es decir, no se admite una editorial distinta a las definidas previamente.
* **Si FK no referencia una tupla**—por ejemplo, el valor del código de departamento de un profesor es nulo—, entonces el profesor no estaría asignado a ningún departamento (válido siempre que este supuesto semántico —sin asignación— tenga cabida en el universo modelado).

La cardinalidad permite indicar la cantidad de elementos relacionados entre sí: N o M indican más de un elemento, 1 indica solo uno y 0 indica que puede haber relación, pero no es obligatoria.

**Diseño de bases de datos relacionales**

Lo importante es encontrar el modo de evaluar adecuadamente la calidad de cada esquema. En este sentido, un mal comportamiento correspondería a esquemas que almacenan información redundante (repetida) o inconsistente y devuelven información contradictoria.

**Dependencias funcionales**

Las dependencias funcionales (DF) son reglas independientes del tiempo que **verifican los atributos de un contexto determinado**. *Independientes del tiempo* quiere decir que estas dependencias se definen y mantienen durante toda la vida útil de la base de datos. Por ejemplo, el atributo *signo del Zodiaco* de una persona depende del atributo *fecha de nacimiento*.

* **Descriptores equivalentes:** Se dice que X e Y son equivalentes cuando por cada valor de X se tiene un valor de Y y por cada valor de Y se tiene un valor de X, y se representa de la siguiente manera: X ⇔ Y. Es importante tener esto en cuenta a la hora de identificar las dependencias funcionales, porque es posible pensar que Y depende de X si no se analiza el sentido inverso (que X dependa o no de Y), de manera que podrían identificarse dos descriptores equivalentes como una dependencia funcional
* Se dice que una dependencia funcional es plena o completa cuando A depende de B, pero no de ningún subconjunto de B.  
  B es un descriptor determinante de A.  
  Ejemplo:  
  ***S# → Ciudad*** es una DF plena o completa porque el valor contenido en *Ciudad* depende directamente del código del proveedor.

**Atributos extraños:** Son los atributos del descriptor de una DF que sobran en la dependencia.  
Ejemplo:  
***(S#, Noms) → Ciudad*** no es una DF plena o completa porque solamente con el código de proveedor *(S#)* ya se determina *Ciudad*. Por lo que Noms es un atributo extraño que no aporta nada en la dependencia.

* **Dependencia funcional elemental:**

Si una dependencia funcional plena tiene como descriptor determinado un atributo simple, la dependencia es elemental.  
***S# → Ciudad*** es una DF elemental ya que *Ciudad* es un atributo simple (no compuesto).

* **Dependencia funcional trivial (reflexividad de Ärmstrong):**

Una DF *X → Y* es trivial si Y es un subconjunto de X; es decir, si X es un conjunto de atributos de la relación e Y es un subconjunto que está contenido en X. Por ejemplo, siendo *ciudad, localidad, comunidad autónoma* y *calle* atributos de X, un conjunto Y con los dos primeros atributos recién mencionados sería una dependencia funcional trivial de X, ya que en realidad Y es un subconjunto de X.

* **Dependencia funcional transitiva:**

Se dice que Z tiene una DF transitiva de X a través de Y si X → Y e Y → Z, pero no Y → X.  
Ejemplo: ***S# → Ciudad → País***. Existe la DF transitiva ***S# → País***o, lo que es lo mismo, con ***S#*** puedes saber la ciudad y con la ciudad el país, pero con el país no puedes saber la ciudad o *S#*.

**Grafo de dependencias funcionales (GDF)**

El grafo de dependencias funcionales es un grafico útil donde se representa el sistema o universo del problema, donde los atributos se representan como nodos y las dependencias se representan como arcos entre los nodos. Además, se pueden hacer agrupaciones de estos nodos dibujando burbujas. Es habitual representar las dependencias desde un nodo o burbuja a un único atributo.  
Sean los atributos *{s#, Noms, Estado, Ciudad, p#, Nomp, Color, Peso, Cant}* y las siguientes dependencias funcionales:  
F = {S# → Noms, S# → Estado, S# → Ciudad, P# →  Nomp, P#  →  Color, P#  → Peso, P#  →  Ciudad, (S#, P#)  →  Cant}  
De ellos, se obtiene el siguiente gráfico de dependencias funcionales:

