1. Modelo de datos.

Según el DRAE, un modelo es, entre otras definiciones, el esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja. Podemos decir que es la representación de cualquier aspecto o tema extraído del mundo real. ¿Qué sería entonces un modelo de datos? Aquél que nos permite describir los elementos que intervienen en una realidad o en un problema dado y la forma en que se relacionan dichos elementos entre sí.

En informática, un **modelo de datos** es un **lenguaje utilizado para la descripción de una base de datos**. Con este lenguaje vamos a poder describir las **estructuras** de los datos (tipos de datos y relaciones entre ellos), las **restricciones de integridad** (condiciones que deben cumplir los datos, según las necesidades de nuestro modelo basado en la realidad) y las operaciones de manipulación de los datos (insertado, borrado, modificación de datos).

Es importante distinguir entre modelo de datos y esquema.

Según Dittrich (1994): *"La descripción específica de un determinado mini-mundo en términos de un modelo de datos se denomina **esquema** (o esquema de datos) del mini-mundo. La colección de datos que representan la información acerca del mini-mundo constituye la base de datos"*

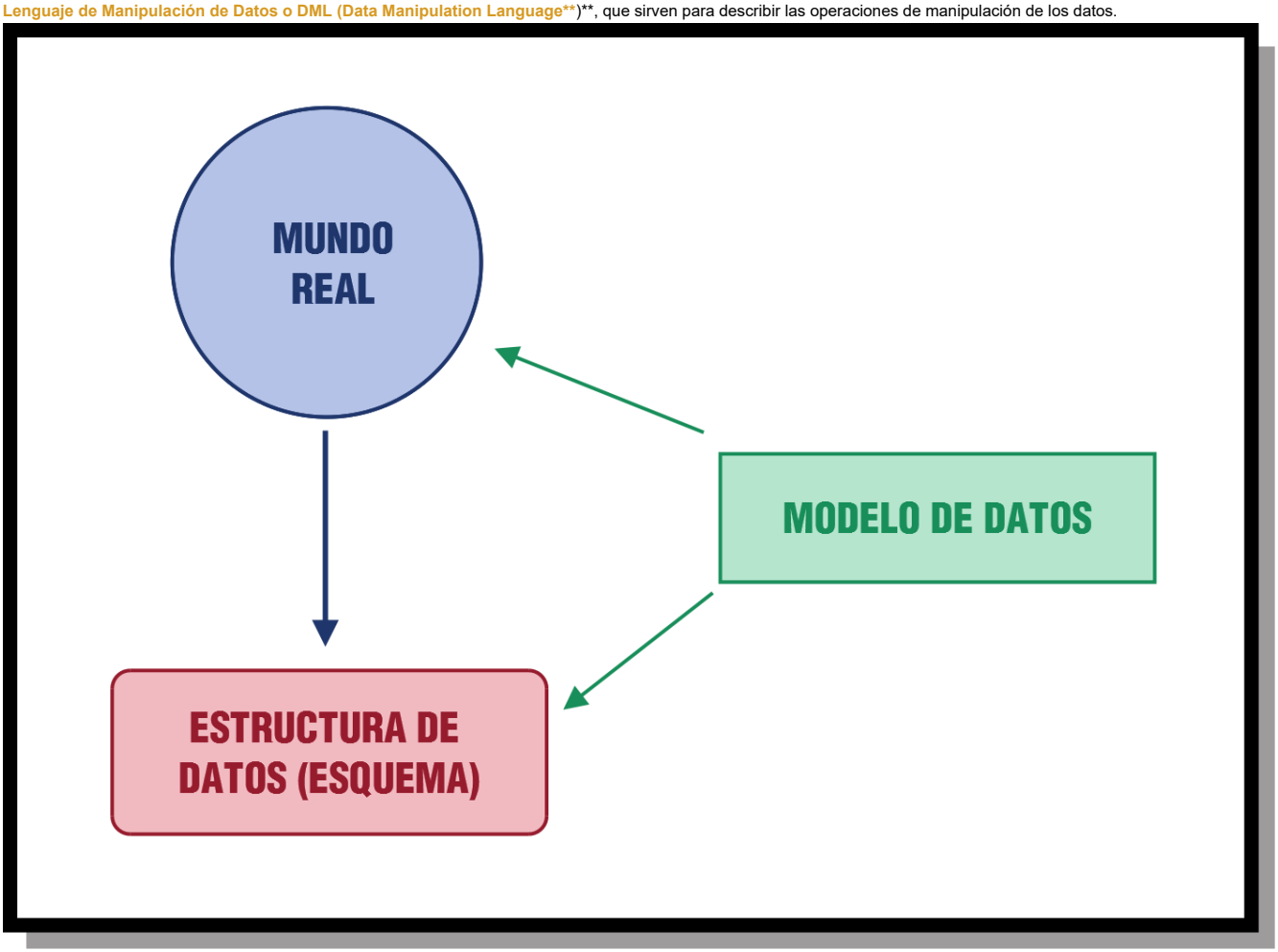
Para De Miguel, Piattini y Marcos (1999) un esquema es la *"representación de un determinado mundo real (universo del discurso) en términos de un modelo de datos".*

Para clasificar los modelos debemos pensar en el nivel de abstracción, es decir, en lo alejado que esté del mundo real:

- Los **modelos de datos conceptuales** son aquellos que describen las estructuras de datos y restricciones de integridad. Se utilizan durante la etapa de análisis de un problema dado, y están orientados a representar los elementos que intervienen y sus relaciones. Ejemplo, Modelo Entidad-Relación.
- Los **modelos de datos lógicos** se centran en las operaciones y se implementan en algún sistema gestor de base de datos. Ejemplo, Modelo Relacional.
- Los **modelos de datos físicos**, son estructuras de datos a bajo nivel, implementadas dentro del propio sistema gestor de base de datos.

Hemos dicho que un modelo de datos es un lenguaje y por lo general, presenta dos sublenguajes:

- **Lenguaje de Definición de Datos o DDL (Data Definition Language**)**, cuya función es describir, de una forma abstracta, las estructuras de datos y las restricciones de integridad.
- **Lenguaje de Manipulación de Datos o DML (Data Manipulation Language**)**, que sirven para describir las operaciones de manipulación de los datos.



2. Terminología del modelo relacional

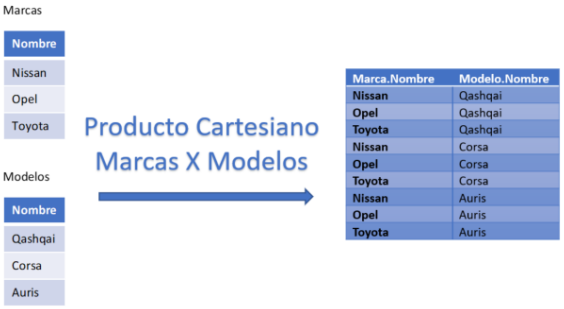
¿Sabes que el modelo relacional te va a permitir representar la información del mundo real de una manera intuitiva? Así es, pudiendo introducir conceptos cotidianos y fáciles de entender por cualquiera, aunque no sea experto en informática.

El modelo relacional fue propuesto por **Edgar Frank Codd** en los laboratorios de **IBM** en California. Como hemos visto, se trata de un modelo lógico que establece una estructura sobre los datos, independientemente del modo en que luego los almacenemos. Es como si guardamos nuestra colección de libros, dependiendo del número de habitaciones que tenga en casa, del tamaño y forma de nuestras estanterías, podremos disponer nuestros libros de un modo u otro para facilitarnos el acceso y consulta. Los libros serán los mismos pero puedo disponerlos de distinta forma.

El nombre de modelo relacional viene de la estrecha relación entre el elemento básico de este modelo y el concepto matemático de relación. Si tenemos dos conjuntos A y B, una relación entre estos dos conjuntos sería un subconjunto del producto cartesiano AxB.

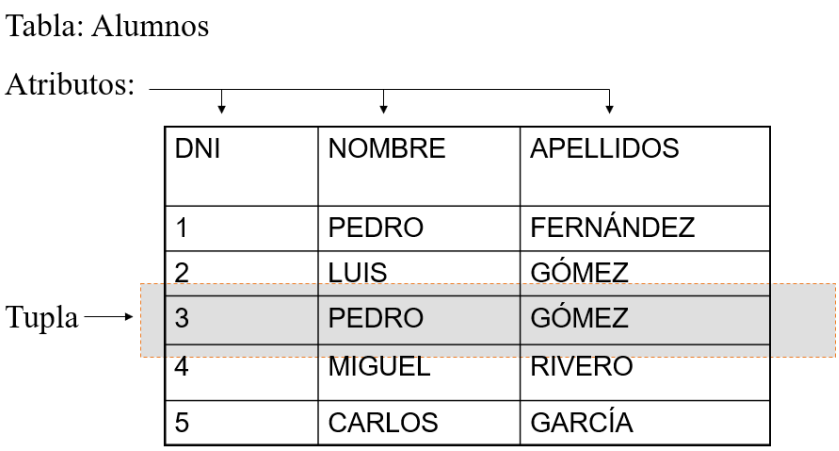
El producto cartesiano nos dará la relación de todos los elementos de un conjunto con todos los elementos de los otros conjuntos de ese producto. Al estar trabajando con conjuntos, no puede haber elementos repetidos.

Por ejemplo para los dos conjuntos que se presentan en la imagen, uno denominado Marcas que contiene marcas de coches y otro denominado Modelos que contiene diferentes modelos el resultado de la operación de producto cartesiano será la combinación de todos los elementos de un conjunto con los del otro.



2.1. Relación o tabla. Tuplas. Dominios.

Pero... ¿qué es eso de "relación"? Hemos dicho que el modelo relacional se basa en el concepto matemático de relación, ya que Codd, que era un experto matemático, utilizó una terminología perteneciente a las matemáticas, en concreto a la teoría de conjuntos y a la lógica de predicados.
A partir de ahora, nosotros veremos una relación como una **tabla** con **filas** y **columnas**. Podemos asociar atributos a columnas y tuplas a filas.



Atributos: es el nombre de cada dato que se almacena en la relación (tabla). Ejemplos serían: DNI, nombre, apellidos, etc.

El nombre del atributo debe describir el significado de la información que representa. En la tabla **Empleados**, el atributo **Sueldo** almacenará el valor en euros del sueldo que recibe cada empleado. A veces es necesario añadir una pequeña descripción para aclarar un poco más el contenido. Por ejemplo, si el sueldo es neto o bruto.

Tuplas: Se refiere a cada elemento de la relación. Si una tabla guarda datos de un cliente, como su DNI o Nombre, una tupla o registro sería ese DNI y nombre concreto de un cliente. Cada una de las filas de la tabla se corresponde con la idea de registro y tiene que cumplir que:

- Cada tupla se debe corresponder con un elemento del mundo real.
- No puede haber dos tuplas iguales (con todos los valores iguales).

Está claro que un atributo en una tupla no puede tomar cualquier valor. No sería lógico que en un atributo Población se guarde "250€". Estaríamos cometiendo un error, para evitar este tipo de situaciones obligaremos a que cada atributo sólo pueda tomar los valores pertenecientes a un conjunto de valores previamente establecidos, es decir, un atributo tiene asociado un **dominio** de valores.

A menudo un dominio se define a través de la declaración de un tipo para el atributo (por ejemplo, diciendo que es un número entero entre 1 y 16), pero también se pueden definir dominios más complejos y precisos. Por ejemplo, para el atributo Sexo de mis usuarios, podemos definir un dominio en el que los valores posibles sean "M" o "F" (masculino o femenino).

Una característica fundamental de los dominios es que sean **atómicos**, es decir, que los valores contenidos en los atributos no se pueden separar en valores de dominios más simples.

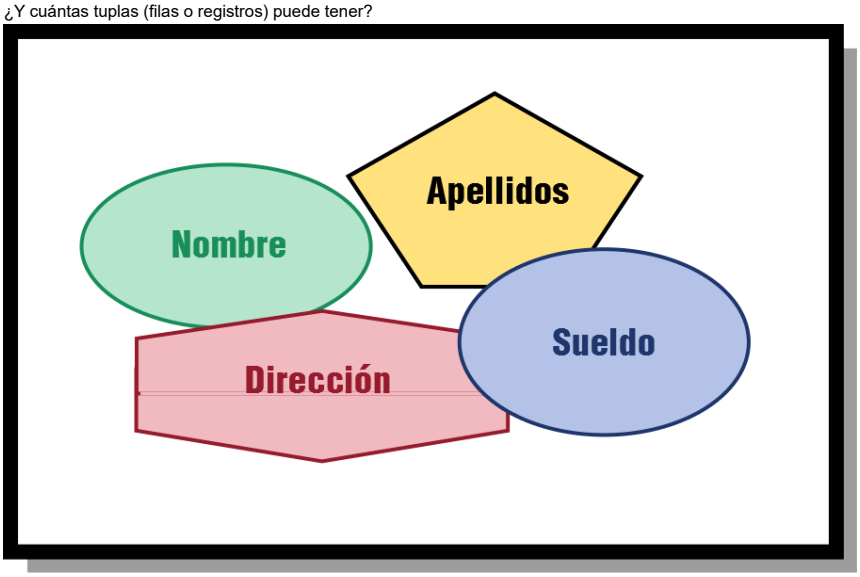
Un dominio debe tener: **Nombre, Definición lógica, Tipo de datos y Formato.**
Por ejemplo, si consideramos el Sueldo de un empleado, tendremos:

- **Nombre**: Sueldo.
- **Definición lógica**: Sueldo neto del empleado
- **Tipo de datos**: número entero.
- **Formato**: 9.999€.

2.2. Grado. Cardinalidad.

Ya hemos visto que una relación es una tabla con filas y columnas. Pero ¿hasta cuántas columnas puede contener? ¿Cuántos atributos podemos guardar en una tabla?
Llamaremos **grado** al tamaño de una tabla en base a su número de atributos (columnas). Mientras mayor sea el grado, mayor será su complejidad para trabajar con ella.

¿Y cuántas tuplas (filas o registros) puede tener?



Llamaremos **cardinalidad** al número de tuplas o filas de una relación o tabla.
Vamos a verlo con un ejemplo. Relación de grado 3, sobre los dominios A={Carlos, María}, B={Matemáticas, Lengua}, C={Aprobado, Suspenso}.
Las posibles relaciones que obtenemos al realizar el producto cartesiano AxBxC es el siguiente:

Producto Cartesiano AxBxC.

A={Carlos, María}	B={Matemáticas, Lengua}	C={Aprobado, Suspenso}
CARLOS	MATEMÁTICAS	APROBADO
CARLOS	MATEMÁTICAS	SUSPENSO
CARLOS	LENGUA	APROBADO
CARLOS	LENGUA	SUSPENSO
CARLOS	INGLÉS	APROBADO
CARLOS	INGLÉS	SUSPENSO
MARÍA	MATEMÁTICAS	APROBADO
MARÍA	MATEMÁTICAS	SUSPENSO
MARÍA	LENGUA	APROBADO
MARÍA	LENGUA	SUSPENSO
MARÍA	INGLÉS	APROBADO
MARÍA	INGLÉS	SUSPENSO

Si cogemos un subconjunto de ésta con 5 filas, tendríamos una relación de cardinalidad 5:

Subconjunto del Producto Cartesiano AxBxC con cardinalidad 5.

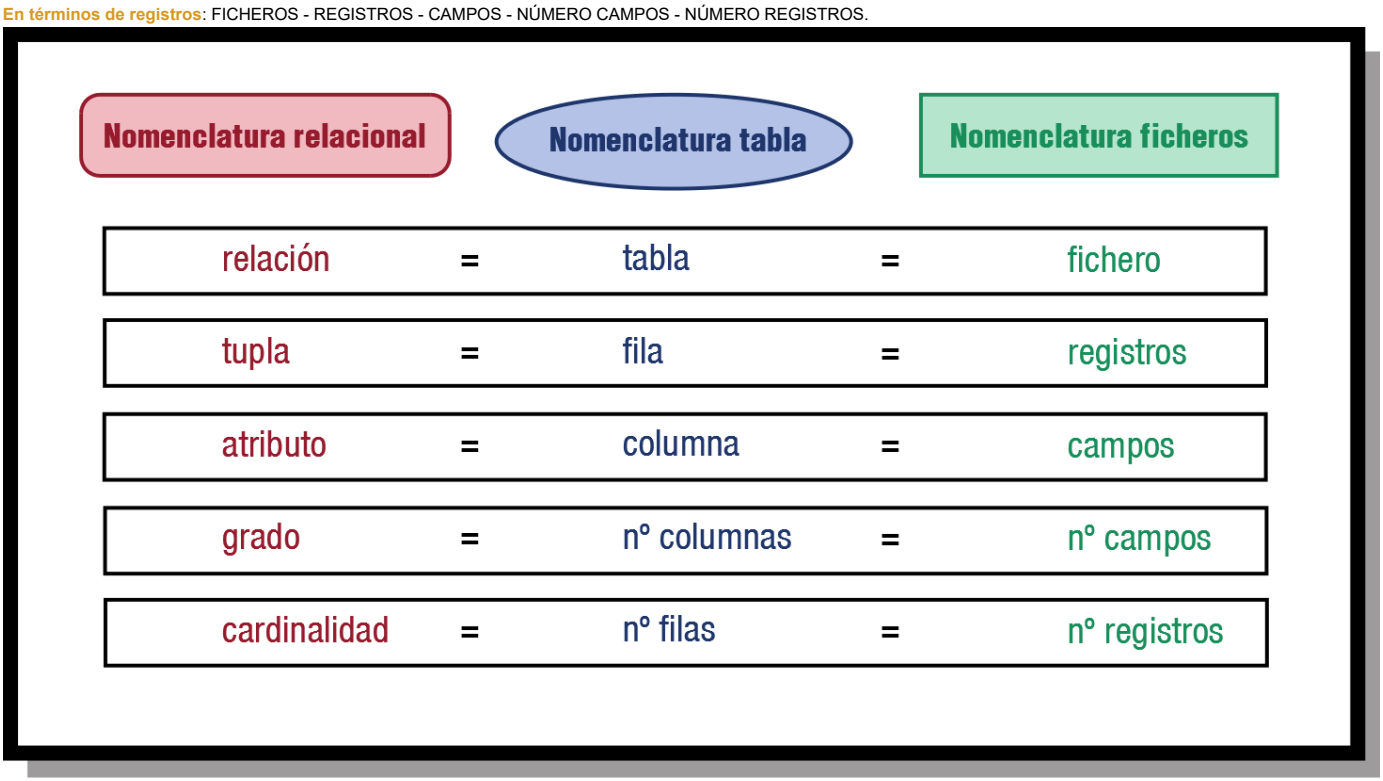
A={Carlos, María}	B={Matemáticas, Lengua}	C={Aprobado, Suspenso}
CARLOS	MATEMÁTICAS	APROBADO
CARLOS	LENGUA	APROBADO
CARLOS	INGLÉS	APROBADO
MARÍA	MATEMÁTICAS	APROBADO
MARÍA	INGLÉS	SUSPENSO

2.3. Sinónimos

Los términos vistos hasta ahora tienen distintos sinónimos según la nomenclatura utilizada. Trabajaremos con tres:

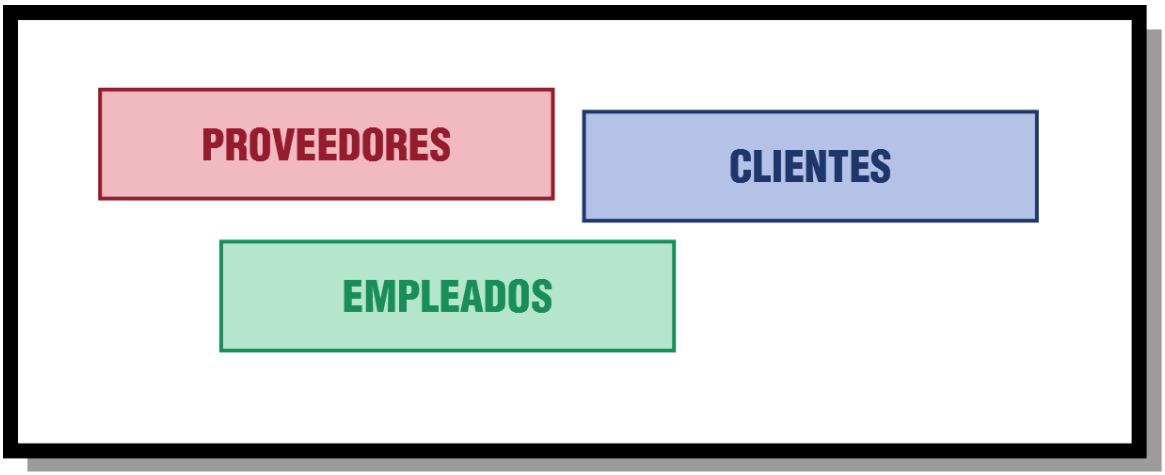
- **En el modelo relacional**: RELACIÓN - TUPLA - ATRIBUTO - GRADO - CARDINALIDAD.
- **En tablas**: TABLA - FILA - COLUMNAS - NÚMERO COLUMNAS - NÚMERO FILAS.

- **En términos de registros**: FICHEROS - REGISTROS - CAMPOS - NÚMERO CAMPOS - NÚMERO REGISTROS.



3. Relaciones. Características de una relación (tabla)

¿En un modelo relacional se puede utilizar cualquier relación? ¿Es válida cualquier tabla o se deben cumplir algunas propiedades?
Debes saber que:
Cada tabla tiene un **nombre distinto**.



Como hemos visto antes, cada atributo (columna) de la tabla **toma un solo valor** en cada tupla (fila).
Cada atributo (columna) tiene un nombre distinto en cada tabla (pero puede ser el mismo en tablas distintas).
No puede haber dos tuplas (filas) **completamente iguales**

Carlos	Matemáticas	Aprobado
Carlos	Matemáticas	Suspenso
Carlos	Lengua	Aprobado
~~Carlos~~	~~Lengua~~	~~Aprobado~~

El **orden** de las tuplas (filas) **no importa**.

a	20
b	30
c	70

=

a	20
c	70
b	30

El **orden** de los atributos (columnas) **no importa**.

a	20
b	30
c	70

=

20	a
70	c
30	b

Todos los datos de un atributo (columna) deben ser del **mismo dominio**.

Carlos	Matemáticas	Aprobado
Carlos	Lengua	Suspenso
Carlos	Inglés	NOTABLE
María	Matemáticas	Suspenso
María	Lengua	Suspenso

4. Tipos de relaciones (tablas)

Existen varios tipos de relaciones y las vamos a clasificar en:

Persistentes: Sólo pueden ser borradas por los usuarios.

- **Base**: Independientes, se crean indicando su estructura y sus ejemplares (conjunto de tuplas o filas).
- **Vistas**: son tablas que sólo almacenan una definición de consulta, resultado de la cual se produce una tabla cuyos datos proceden de las bases o de otras vistas e instantáneas. Si los datos de las tablas base cambian, los de la vista que utilizan esos datos también cambiarán.
- **Instantáneas**: son vistas (se crean de la misma forma) que sí almacenan los datos que muestran, además de la consulta que la creó. Solo modifican su resultado cuando el sistema se refresca cada cierto tiempo. Es como una fotografía de la relación, que sólo es válida durante un periodo de tiempo concreto.

Temporales: Son tablas que son eliminadas automáticamente por el sistema.

5. Tipos de datos

¿Qué es un DNI? ¿Con qué datos lo representamos? DNI es una información que es susceptible de ser guardada. Normalmente el DNI está formado por dígitos y una letra al final. Si tuviéramos que clasificarlo diríamos que es un conjunto de caracteres alfanuméricos. ¿Y si pensamos en Sueldo? Aquí lo tenemos un poco más claro, evidentemente es un número entero o con decimales.

Hasta ahora hemos visto que vamos a guardar información relacionada en forma de filas y columnas. Las columnas son los atributos o información que nos interesa incluir del mundo real que estamos modelando.

Hemos visto que esos atributos se mueven dentro de un dominio, que formalmente es un conjunto de valores. Pues bien, en términos de sistemas de base de datos, se habla más de tipos de datos que de dominios. Al crear la relación (tabla) decidimos qué conjunto de datos deberá ser almacenado en las filas de los atributos que hemos considerado. Tenemos que asignar un tipo de dato a cada atributo.

Con la asignación de tipos de datos, también habremos seleccionado un dominio para un atributo.
Cada campo:

- debe poseer un **Nombre** (relacionado con los datos que va a contener) y
- debe tener asociado un **Tipo de dato.**

Existen distintas formas de nombrar los tipos de datos dependiendo del lenguaje que utilicemos (C, Java, PHP, MySQL, SQL, Pascal, etc.).

Veamos cuales son los tipos de datos más comunes con los que nos encontraremos generalmente:

- **Texto**: almacena cadenas de caracteres (números con los que **no** vamos a realizar operaciones matemáticas, letras o símbolos).
- **Numérico**: almacena números con los que vamos a realizar operaciones matemáticas.
- **Fecha/hora**: almacena fechas y horas.
- **Sí/No**: almacena datos que solo tienen dos posibilidades (verdadero/falso).
- **Autonumérico**: valor numérico secuencial que el SGBD incrementa de modo automático al añadir un registro (fila).
- **Memo**: almacena texto largo (mayor que un tipo texto).
- **Moneda**: se puede considerar un subtipo de Numérico ya que almacena números, pero con una característica especial, y es que los valores representan cantidades de dinero.
- **Objeto OLE**: almacena gráficos, imágenes o textos creados por otras aplicaciones.

Para saber más

Si quieres saber un poco más sobre los tipos de datos puedes ver estos enlaces
Wikipedia tipos de datos - Tipos de datos
Oracle Data Types - Database SQL Language Reference

6. Claves

¿Cómo diferenciamos unos usuarios de otros? ¿Cómo sabemos que no estamos recogiendo la misma información? ¿Cómo vamos a distinguir unas tuplas de otras? Lo haremos mediante los valores de sus atributos. Para ello, buscaremos un atributo o un conjunto de atributos que identifiquen de modo único las tuplas (filas) de una relación (tabla). A ese atributo o conjunto de atributos lo llamaremos **superclaves**.

Hemos visto que una característica de las tablas era que no puede haber dos tuplas (filas) completamente iguales, con lo que podemos decir que toda la fila como conjunto sería una superclave.

Por ejemplo, en la tabla Usuarios tenemos las siguientes superclaves:

- {Nombre, Apellidos, login, e_mail, F_nacimiento}
- {Nombre, Apellidos, login, e_mail}
- {login, e_mail}
- {login}

Tendríamos que elegir alguna de las superclaves para diferenciar las tuplas. En el modelo relacional trabajamos con tres tipos de claves:

- Claves **candidatas**.
- Claves **primarias**.
- Claves **alternativas.**
- Claves **ajenas**.

A continuación veremos cada una de ellas.

Para saber más

En este enlace tienes más información sobre las claves y superclaves: explicación y ejemplos

6.1. Clave candidata. Clave primaria. Clave alternativa.

Si puedo elegir entre tantas claves, ¿con cuál me quedo? Tendremos que elegir entre las claves "candidatas" la que mejor se adapte a mis necesidades. ¿Y cuáles son éstas? Las claves **candidatas** serán aquel conjunto de atributos que identifiquen de manera única cada tupla (fila) de la relación (tabla). Es decir, las columnas cuyos valores no se repiten en ninguna otra fila de la tabla. Por tanto, cada tabla debe tener **al menos una clave candidata** aunque puede haber más de una.

Siguiendo con nuestro ejemplo, podríamos considerar los atributos Login o E_mail como claves candidatas, ya que sabemos que el Login debe ser único para cada usuario, a E_mail le sucede lo mismo. Pero también cabe la posibilidad de tomar: Nombre, Apellidos y F_nacimiento, las tres juntas como clave candidata.

Las claves candidatas pueden estar formadas por más de un atributo, siempre y cuando éstos identifiquen de forma única a la fila. Cuando una clave candidata está formada por más de un atributo, se dice que es una **clave compuesta**.

Una clave **candidata** debe cumplir los siguientes requisitos:

- **Unicidad**: no puede haber dos tuplas (filas) con los mismos valores para esos atributos.
- **Irreducibilidad**: si se elimina alguno de los atributos deja de ser única.

Si elegimos como clave candidata **Nombre, Apellidos y F_nacimiento**, cumple con la unicidad puesto que es muy difícil encontrarnos con dos personas que tengan el mismo nombre, apellidos y fecha de nacimiento iguales. Es irreducible puesto que sería posible encontrar dos personas con el mismo nombre y apellidos o con el mismo nombre y fecha de nacimiento, por lo que son necesarios los tres atributos (campos) para formar la clave.

Para identificar las claves candidatas de una relación no nos fijaremos en un momento concreto en el que vemos una base de datos. Puede ocurrir que en ese momento no haya duplicados para un atributo o conjunto de atributos, pero esto no garantiza que se puedan producir. El único modo de identificar las claves candidatas es conociendo el significado real de los atributos (campos), ya que así podremos saber si es posible que aparezcan duplicados. Es posible desechar claves como candidatas fijándonos en los posibles valores que podemos llegar a tener. Por ejemplo, podríamos pensar que Nombre y Apellidos podrían ser una clave candidata, pero ya sabemos que cabe la posibilidad de que dos personas puedan tener el mismo Nombre y Apellidos, así que lo descartamos.

Hasta ahora, seguimos teniendo varias claves con la que identificamos de modo único nuestra relación. De ahí el nombre de candidatas. Hemos de quedarnos con una.

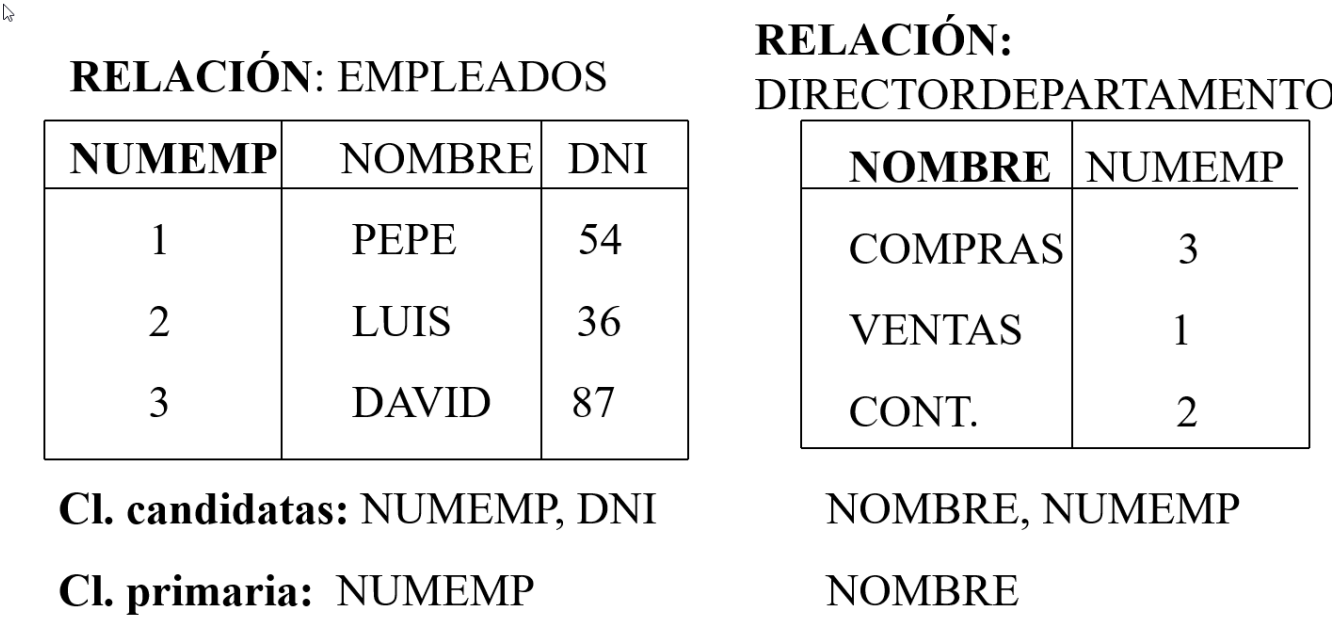
La **clave primaria** de un relación es aquella clave candidata que se escoge para identificar sus tuplas de modo único. Ya que una relación no tiene tuplas duplicadas, siempre hay una clave candidata y, por lo tanto, la relación siempre tiene clave primaria. En el peor caso, la clave primaria estará formada por todos los atributos de la relación, pero normalmente habrá un pequeño subconjunto de los atributos que haga esta función. En otros casos, podemos crear un campo único que identifique las tuplas, por ejemplo un código de usuario, que podrían estar constituidos por valores autonuméricos.

Las claves candidatas que no son escogidas como clave primaria son denominadas **claves alternativas**.

Si en nuestra tabla Usuarios escogemos Login como clave primaria, el E_mail o {Nombre, Apellidos, F_Nacimiento} serán nuestras claves alternativas.

Ejemplo

En la siguiente imagen observamos dos relaciones, Empleados con atributos NumEmp, Nombre y DNI y la relación DirectorDepartamento con atributos Nombre y NumEmp. La primera relación contiene todos los empleados de la empresa, la segunda contiene, para cada departamento, el número de empleado que ejerce las funciones de director. Podemos observar que en cada una de ellas encontramos claves candidatas y entre ellas se selecciona la clave primaria que se considera más adecuada.



6.2. Clave externa, ajena o secundaria

Hasta ahora no nos hemos planteado cómo se relacionan unas tablas con otras dentro de una base de datos. Si tenemos las tablas Usuarios y Partidas, necesariamente habrá una "relación" entre ellas. Deben compartir algún dato en común que las relacione. Una partida es jugada por un jugador (Usuarios), por lo que en la tabla Partida deberíamos guardar algún dato del usuario-jugador, pero ¿cuál?

Una clave **ajena, también llamada externa o secundaria,** es un atributo o conjunto de atributos de una relación cuyos valores coinciden con los valores de la clave primaria de alguna otra relación (o de la misma). Las claves ajenas **representan relaciones entre datos**. Dicho de otra manera, son los datos de atributos de una tabla cuyos valores están relacionados con atributos de otra tabla.

En la tabla Partidas, se recogen datos como **Cod_partida**, **Fecha** y **Hora de creación**, **Nombre de la partida**, etc. ¿Qué campo utilizaremos para relacionarla con la tabla Usuarios? Si nos basamos en la definición, deberíamos utilizar la clave primaria de la tabla Usuarios. Por tanto, el atributo Login que es la clave principal en su tabla aparecerá en la tabla Partidas como clave ajena, externa o secundaria. El Login en Partidas hace referencia a cada jugador que juega esa partida. En lugar de guardar todos los datos de ese jugador en la misma tabla, lo hacemos en otra y lo "referenciamos" por su clave primaria tomándola como ajena.

Es lógico que las claves ajenas no tengan las mismas propiedades y restricciones que tienen como clave primaria en su tabla, por tanto, sí que pueden repetirse en la tabla. En nuestro ejemplo, un mismo jugador puede jugar varias partidas.

Las claves ajenas tienen por objetivo establecer una conexión con la clave primaria que referencian. Por lo tanto, los valores de una clave ajena deben estar presentes en la clave primaria correspondiente, o bien deben ser valores nulos. En caso contrario, la clave ajena representaría una referencia o conexión incorrecta.

No podemos tener una partida de un jugador que previamente no se ha registrado. Pero sí podemos tener los datos de una partida y desconocer el jugador de ésta.

Ejemplo

Entre las dos relaciones del ejemplo anterior existe una clave ajena, en este caso es el campo NumEmp de la relación DirectorDepartamento y que asegura que no puede existir un empleado dirigiendo un departamento de la empresa que no se encuentre en la relación empleados.

RELACIÓN: EMPLEADOS

NUMEMP	NOMBRE	DNI
1	PEPE	54
2	LUIS	36
3	DAVID	87

RELACIÓN: DIRECTORDEPARTAMENTO

NOMBRE	NUMEMP
COMPRAS	3
VENTAS	1
CONT.	2

Cl. ajena: NUMEMP en DIRECTORESDEPARTAMENTO es clave ajena sobre la relación EMPLEADOS

7. Índices

Imagina que estás creando un diccionario de términos informáticos. Podrías elegir la opción de escribirlo en una única hoja muy larga (estilo pergamino) o bien distribuirlo por hojas. Está claro que lo mejor sería distribuirlo por páginas. Y si buscamos el término "informática" en nuestro diccionario, podríamos comenzar a buscar en la primera página y continuar una por una hasta llegar a la palabra correspondiente. O bien crear un índice al principio, de manera que podamos consultar a partir de qué página podemos localizar las palabras que comienzan por "i". Esta última opción parece la más lógica.

Pues bien, en las bases de datos, cada tabla se divide internamente en páginas de datos, y se define el índice a través de un campo (o campos) y es a partir de este campo desde donde de busca.

Un **índice** es una estructura de datos que permite acceder a diferentes filas de una misma tabla a través de un campo o campos . Esto permite un acceso mucho más rápido a los datos.

Los índices son útiles cuando se realizan consultas frecuentes a un rango de filas o una fila de una tabla. Por ejemplo, si consultamos los usuarios cuya fecha de ingreso es anterior a una fecha concreta.

Los cambios en los datos de las tablas (agregar, actualizar o borrar filas) son incorporados automáticamente a los índices con transparencia total.

Debes saber que los índices son independientes, lógica y físicamente de los datos, es por eso que pueden ser creados y eliminados en cualquier momento, sin afectar a las tablas ni a otros índices.

¿Cuándo indexamos? No hay un límite de columnas a indexar, si quisiéramos podríamos crear un índice para cada columna, pero no sería operativo. Normalmente tiene sentido crear índices para ciertas columnas ya que agilizan las operaciones de búsqueda de base de datos grandes. Por ejemplo, si la información de nuestra tabla Usuarios se desea consultar por apellidos, tiene sentido indexar por esa columna.

Al crear índices, las operaciones de modificar o agregar datos se ralentizan, ya que al realizarlas es necesario actualizar tanto la tabla como el índice.

Si se elimina un índice, el acceso a datos puede ser más lento a partir de ese momento.

8. El valor Nulo (NULL). Operaciones con ese valor

¿Qué sucede si al guardar los datos de los Usuarios hay algún dato que no tengo o no necesito guardarlo porque no corresponde?

Independientemente del dominio al que pertenezca un campo, éste puede tomar un valor especial denominado **NULO** (**NULL** en inglés) que designará la ausencia de dato.

Cuando por cualquier motivo se desconoce el valor de un campo, por ejemplo, desconocemos el teléfono del usuario, o bien ese campo carece de sentido (siguiendo con el mismo ejemplo, puede que el usuario no tenga teléfono), podemos asignar a ese campo el valor especial NULO.

Cuando trabajamos con claves secundarias el valor nulo indica que la tupla o fila no está relacionada con ninguna otra tupla o fila. Este valor NULO es común a cualquier dominio.

Pero ten en cuenta una cosa, no es lo mismo valor NULO que ESPACIO EN BLANCO.

Tampoco será lo mismo valor NULO que el valor CERO.

Un ordenador tomará un espacio en blanco como un carácter como otro cualquiera. Por tanto, si introducimos el carácter "espacio en blanco" estaríamos introduciendo un valor que pertenecería al dominio **texto** y sería distinto al concepto "ausencia de valor" que sería **no incluir nada** (nulo).

Este valor se va a utilizar con frecuencia en las bases de datos y es imprescindible saber cómo actúa cuando se emplean operaciones lógicas sobre ese valor. En la [lógica booleana](https://fpdistancia.educa.madrid.org/mod/glossary/showentry.php?displayformat=dictionary&concept=L%C3%B3gica%20booleana%20%28DAW_BD02%29 "Ver la definición de "Lógica booleana" (Se abre en una nueva ventana)") tenemos los valores VERDADERO y FALSO, pero un valor NULO no es ni verdadero ni falso.

Cuando necesitemos comparar dos campos, si ambos son nulos no podremos obtener ni verdadero ni falso. Necesitaremos definir la lógica con este valor. Veamos los operadores lógicos más comunes y sus resultados utilizando el valor nulo:

- VERDADERO Y (AND) NULO daría como resultado NULO.
- FALSO Y (AND) NULO daría como resultado FALSO.
- VERDADERO O (OR) NULO daría como resultado VERDADERO.
- FALSO O NULO daría como resultado NULO.
- NO (NOT) NULO daría como resultado NULO.

En todas las bases de datos relacionales se utiliza un operador llamado ES NULO (IS NULL) que devuelve VERDADERO si el valor con el que se compara es NULO.

Para saber más

El uso del valor nulo es un tema que da mucho que hablar:

Wikipedia - Valores nulos

Jennifer Null es la demostración de cómo tu nombre puede convertirse en tu peor pesadilla en Internet (Fuente: Xataka.com)

9. Vistas

Cuando vimos los distintos tipos de relaciones, aprendimos que, entre otros, estaban las vistas. Ahora ya tenemos más conocimientos para comprender mejor este concepto.

Una **vista** es una tabla "virtual" cuyas filas y columnas se obtienen a partir de una o de varias tablas que constituyen nuestro modelo. Lo que se almacena no es la tabla en sí, sino su definición, por eso decimos que es "virtual". Una vista actúa como filtro de las tablas a las que hace referencia en ella.

La consulta que define la vista puede provenir de una o de varias tablas, o bien de otras vistas de la base de datos actual u otras bases de datos.

No existe ninguna restricción a la hora de consultar vistas y muy pocas restricciones a la hora de modificar los datos de éstas.

Podemos dar dos razones por las que queramos crear vistas:

- **Seguridad,** nos puede interesar que los usuarios tengan acceso a una parte de la información que hay en una tabla, pero no a toda la tabla.
- **Comodidad**, como veremos al pasar nuestras tablas/relaciones a un lenguaje de base de datos, puede que tengamos que escribir sentencias bastante complejas, las vistas no son tan complejas.

Las vistas no tienen una copia física de los datos, son consultas a los datos que hay en las tablas, por lo que si actualizamos los datos de una vista, estamos actualizando realmente la tabla, y si actualizamos la tabla estos cambios serán visibles desde la vista.

Aunque **no siempre** podremos actualizar los datos de una vista, dependerá de la complejidad de la misma y del gestor de base de datos. No todos los gestores de bases de datos permiten actualizar vistas, Oracle y SQL Server, por ejemplo, lo permiten.

10. Usuarios. Roles. Privilegios.

A la hora de conectarnos a la base de datos es necesario que utilicemos un modo de acceso, de manera que queden descritos los permisos de que dispondremos durante nuestra conexión. En función del nombre de usuario tendremos unos permisos u otros.

Un **usuario** es un conjunto de permisos que se aplican a una conexión de base de datos. Tiene además otras funciones como son:

- Ser el propietario de ciertos objetos (tablas, vistas, etc.).
- Realiza las copias de seguridad.
- Define una cuota de almacenamiento.
- Define el tablespace por defecto para los objetos de un usuario en Oracle.

Pero no todos los usuarios deberían poder hacer lo mismo cuando acceden a la base de datos. Por ejemplo, un administrador debería tener más privilegios que un usuario que quiere realizar una simple consulta.

¿Qué es un **privilegio**? No es más que un permiso dado a un usuario para que realice ciertas operaciones, que pueden ser de dos tipos:

- **De sistema**: necesitará el permiso de sistema correspondiente.
- **Sobre objeto**: necesitará el permiso sobre el objeto en cuestión.

¿Y no sería interesante poder agrupar esos permisos para darlos juntos? Para eso tenemos el rol.

Un **rol** de base de datos no es más que una agrupación de permisos de sistema y de objeto.

Podemos tener a un grupo determinado de usuarios que tengan permiso para consultar los datos de una tabla concreta y no tener permiso para actualizarlos. Luego un rol permite asignar un grupo de permisos a un usuario. De este modo, si asignamos un rol con 5 permisos a 200 usuarios y luego queremos añadir un permiso nuevo al rol, no tendremos que ir añadiendo este nuevo permiso a los 200 usuarios, ya que el rol se encarga de propagarlo automáticamente.