UD 02: Introducción al modelo relacional

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.

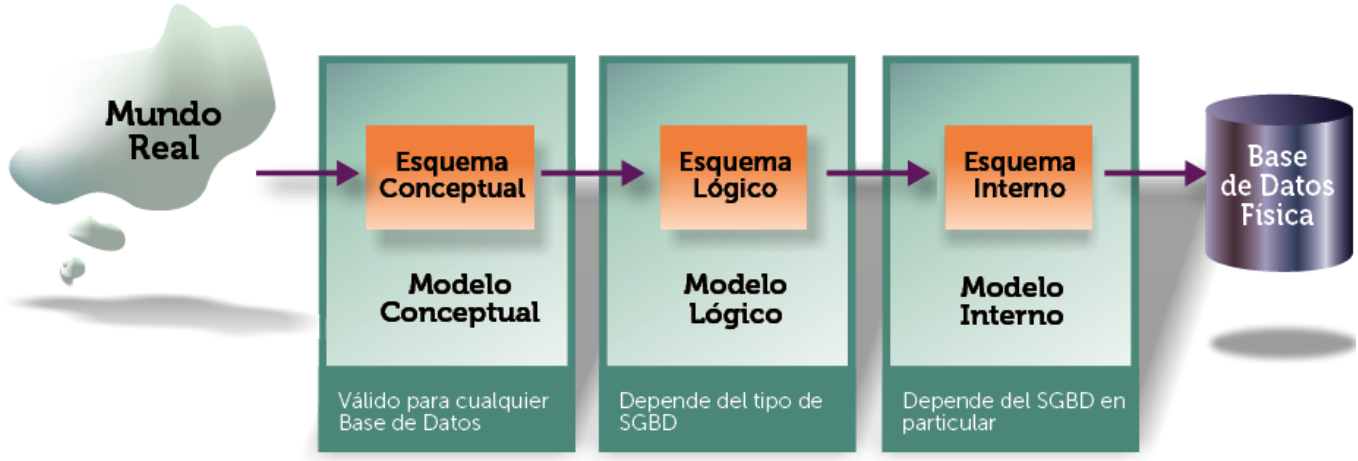


Imagen: diferentes tipos de modelos según su nivel de abstracción.

Según el nivel de abstracción, podemos clasificar diferentes modelos:

- **Modelos de datos conceptuales**: son aquellos que describen las estructuras de datos y restricciones de integridad. Se utilizan durante la etapa del análisis de un problema dad, y están orientados a representar los elementos que intervienen y sus asociaciones. Por ejemplo, el modelo entidad-relación.
- **Modelos de datos lógicos**: se centran en las operaciones y se implementan por un tipo de sistema gestor de base de datos. Por ejemplo, el **modelo relacional**.
- **Modelos de datos físicos**: son estructuras de datos de bajo nivel, implementadas por un sistema gestor de base de datos concreto.

> Es decir, un mismo modelo conceptual se podría representar mediante un modelo relacional o mediante un modelo NoSQL.
>
> Por otro lado, un mismo modelo lógico se representará con diferentes modelos físicos si lo implementamos con MySQL o con Postgresql.

2. El modelo relacional

2.1 Introducción

Durante los primeros años de las bases de datos, se utilizó el **modelo jerárquico** como la primera forma de describir de forma más humana una base de datos. Después reinó el **modelo en red** especialmente en su norma **Codasyl**. Así a principios de los 70 parecía que el modelo a aplicar al implementar bases de datos sería Codasyl y lo sería por muchos años.

Sin embargo, **Edgar Frank Codd** definió las bases del modelo relacional a finales de los 60. En 1970 publica el documento *"A Relational Model of data for Large Shared Data Banks"* (*"Un modelo relacional de datos para grandes bancos de datos compartidos"*). Actualmente se considera que ese es uno de los documentos más influyentes de toda la historia de la informática. Lo es porque en él se definieron las bases del llamado **Modelo**

Relacional de Bases de Datos. Anteriormente el único modelo teórico estandarizado era el modelo Codasyl que se utilizó masivamente en los años 70 como paradigma del modelo en red de bases de datos.

Codd se apoya en los trabajos de los matemáticos **Cantor** y **Childs** (cuya teoría de conjuntos es la verdadera base del modelo relacional). Según Codd, los datos se agrupan en **relaciones** (actualmente llamadas **tablas**), las cuales son una estructura que aglutina datos referidos a una misma entidad de forma organizada. Las relaciones, además, estructuran los datos de forma independiente respecto a su almacenamiento real en la computadora. Es decir, es un elemento conceptual, no físico.

Lo que Codd intentaba fundamentalmente es evitar que las usuarias y usuarios de la base de datos tuvieran que verse obligadas a aprender los entresijos internos del sistema. Esto es lo que ocurría con el modelo en red, dominante cuando Codd diseñó el modelo relacional, que era bastante físico. Su enfoque fue revolucionario al evitar conceptos del mundo de la computación en su modelo.

Aunque trabajaba para **IBM**, esta empresa no recibió de buen grado sus teorías. De hecho, IBM continuó trabajando en su sistema gestor de bases de datos en red **IMS**. Fueron otras empresas (en especial **Oracle**) las que implementaron sus teorías.

Pocos años después, el modelo se empezó a utilizar cada vez más hasta, finalmente, ser el modelo de bases de datos más popular. Hoy en día casi todas las bases de datos siguen este modelo, aunque en estos años han aparecido rivales cada vez más fuertes en las llamadas bases de datos **NoSQL**, que han demostrado un gran eficacia en bases de datos que necesitan una enorme cantidad de instrucciones de modificación por minuto.

2.2. Objetivos

Codd perseguía estos objetivos con su modelo relacional:

- **Independencia física**. La forma de almacenar los datos, debe ser absolutamente independiente del modelo conceptual de los mismos. Si la forma de almacenar los datos cambia (si cambia el esquema físico) , no es necesario cambiar los esquemas lógicos, funcionan perfectamente. Esto permite que los usuarios y usuarias se concentren en qué resultados desean obtener de la base de datos independientemente de cómo estén realmente almacenados los datos.
- **Independencia lógica**. Se refiere a que la lógica de la base de datos debe de ser independiente de la forma externa de acceso a la base de datos (los esquemas externos). Las aplicaciones que utilizan la base de datos no deben ser modificadas porque se modifique el esquema lógico de la misma. De una manera más precisa: gracias a esta independencia el esquema externo de la base de datos es realmente independiente del modelo lógico. En la práctica, esta independencia es difícil de conseguir.
- **Flexibilidad**. La base de datos ofrece fácilmente distintas vistas en función de los usuarios y aplicaciones. La visión de los datos se adapta al usuario que los requiere.
- **Uniformidad**. Las estructuras lógicas siempre tienen una única forma lógica (las tablas). Es decir, manejar el modelo relacional es manejar las tablas. Sencillez. Facilidad de manejo (algo cuestionable, pero ciertamente verdadero si comparamos con los sistemas gestores de bases de datos anteriores a este modelo).

2.3 Algo de historia

- En **1970** Codd publica las bases del modelo relacional.
- En **1979** aparece Oracle, el primer Sistema Gestor de Bases de Datos Relacional (SGBDR) comercial. Se convertirá en el lider del mercado.

- En **1986,1987**, ANSI e ISO normalizan el lenguaje SQL.
- En **1988** la versión 6 de Oracle implementa PL/SQL.
- En **1992** ISO publica el estándar SQL 92 (uno de los más utilizados a día de hoy, casi 30 años después).
- En **1995** y **1996** aparecen Mysql y Postgresql, respectivamente.
- En **1999** se actualiza el estándar SQL 99 (también conocido como SQL 2000).
- En **2003**, **2006**, **2008**, **2011** y **2016** se actualiza de nuevo el estándar ISO, dando lugar a sus correspondientes versiones.

> Si quieres conocer algunas diferencias sobre las diferentes versiones del estándar, puedes visitar este enlace: https://es.wikipedia.org/wiki/SQL

3. Estructuras de una base de datos relacional

3.1 Relación o tabla

Según el modelo relacional (desde que Codd lo enunció) el elemento fundamental del modelo es lo que se conoce como **relación**, aunque más habitualmente se le llama **tabla** (o también array o matriz). Codd definió el significado de las relaciones utilizando un lenguaje matemático. Para comprender visualmente este concepto siempre se han utilizado las tablas, ya que permiten representar la información de las relaciones en forma de filas y columnas.

> No hay que confundir la idea de relación según el modelo de Codd, con lo que significa una relación en el modelo Entidad/Relación de Chen. No tienen nada que ver

Las relaciones constan de:

- **Atributos**. Es cada una de las propiedades de los datos de la relación (nombre, dni,...). Las relaciones representan conjuntos de objetos o elementos reales, cada atributo es propiedad o característica de dicho elemento.
- **Tuplas**. Referido a cada ejemplar, objeto o elemento de la relación. Por ejemplo si una relación almacena personas, una tupla representaría a una persona en concreto.

atributo 1	atributo 2	atributo3	....	atributo N	
valor 1,1	valor 1,2	valor 1,3	...	valor 1,N	← tupla 1
valor 2,1	valor 2,2	valor 2,3	...	valor 2, N	← tupla 2
...	...	...	...	...	
valor m,1	valor m,2	valor m,3	....	valor m, N	← tupla m

Imagen: este sería un ejemplo genérico de relación según el modelo de Codd.

La siguiente imagen sería el ejemplo de una relación que representa algunos datos de los alumnos de una clase:

num_alum	nombre	apellido1	apellido2	edad
1	José	García	Muñoz	18
2	María	Alcántara	Leiva	19
3	Miguel	De la Cuadra	Alejo	23
4	Ángela	Antúnez	Almagro	19
5	Carlos	Martínez	Castro	20
6	Lucía	Sierra	Pérez	18

Imagen: ejemplo concreto de una relación según el modelo de Codd.

> La forma de representar relaciones es mediante tablas regulares en las que las columnas se corresponden con los atributos y las filas con las tuplas. Esta forma es más visual y entendible.

3.2 Tupla

Se llama tupla a cada uno de los elementos de una relación. Hablando en términos de tabla, una tupla es una fila.

> Si una tabla guarda datos de un alumno, como su DNI o Nombre, una tupla o registro sería ese DNI y nombre concreto de un alumno.

Las tuplas, en el modelo relacional, cumplen estas premisas:

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

3.3 Dominio

Un dominio contiene todos los posibles valores que puede tomar un determinado atributo. Dos atributos distintos pueden tener el mismo dominio.

Un dominio esta formado por un conjunto finito de valores del mismo tipo. A los dominios se les asigna un nombre y así podemos referirnos a ese nombre en más de un atributo, facilitando la aplicación de los mismos.

> Es decir, un dominio expresa de forma inequívoca los valores posibles dentro de un atributo.

Se confunde la idea entre tipo de datos y dominio. Lo cierto es que son conceptos semejantes, pero no iguales. La diferencia es que el dominio impone más restricciones que los tipos de datos.

Por ejemplo, el texto *Hola*, está claro que no puede ser un valor para el atributo *fecha_nac*, porque no es una fecha. Sin embargo si tuviéramos un atributo llamado *país* el texto Hola tampoco forma parte de los valores posibles para ese atributo, porque *Hola* no es un *país*; y, sin embargo, tanto la expresión *Hola* como el nombre de los países pertenecen al mismo tipo de datos (son textos).

Otro ejemplo de dominio: el que hace que un valor para el atributo *dni* sólo se considere perteneciente a su dominio si tiene ocho números, una letra y además la letra cumple una regla matemática concreta sobre los números.

> ¿Cómo se calcula la letra del DNI?
> https://es.wikipedia.org/wiki/N%C3%BAmero_de_identificaci%C3%B3n_fiscal

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 positivo.
- **Formato**: 9.999€.

La forma de indicar un dominio se puede hacer utilizando dos posibles técnicas:

- **Intensión**. Se define el nomino indicando la definición exacta de sus posibles valores. Por intensión se puede definir el dominio de edades de los trabajadores como: números enteros entre el 16 y el 65 (un trabajador sólo podría tener una edad entre 16 y 65 años).
- **Extensión**. Se indican algunos valores y se sobreentiende el resto gracias a que se autodefinen con los anteriores. Por ejemplo el dominio localidad se podría definir por extensión así: Sevilla, Carmona, San Juan de Aznalfarache,...

Además pueden ser:

- **Generales**. Los valores están comprendidos entre un máximo y un mínimo y pueden tomar cualquier valor intermedio.
- **Restringidos**. Solo pueden tomar un conjunto de valores.

3.4 Grado y 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.

> La mayoría de SGBDR del mercado tienen un límite con respecto al número de columnas. Por ejemplo, Oracle impone, a partir de la versión 8, un límite de 1000 columnas por tabla. Postgresql tiene un máximo de 250 a 1600, dependiendo el tipo de dato de las columnas.

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

Llamaremos **cardinalidad** al número de tuplas o filas de una relación o tabla.

> En general, la mayoría de SGBDR comerciales ampliamente utilizados no tienen un límite de filas por tabla. El límite de almacenamiento viene marcado, más bien, por el de la capacidad de disco subyacente.

3.5 Sinónimos

Los términos vistos anteriormente tienen distintos sinónimos según la nomenclatura utilizada. A ese respecto se utilizan tres nomenclaturas:

Nomenclatura relacional	Nomenclatura visual	Nomenclatura ficheros
relación	tabla	fichero
tupla	fila	registro
atributo	columna	campo
grado	nº columnas	nº campos
cardinalidad	nº filas	nº registros

3.6 Propiedades de las relaciones del modelo relacional

- Cada tabla debe tener un nombre distinto
- Cada atributo de la tabla solo puede tener un valor en cada tupla
- Cada atributo tiene un nombre distinto en cada tabla (aunque puede coincidir en tablas distintas)
- Cada tupla es única (no hay tuplas duplicadas)
- El orden de los atributos no importa
- El orden de las tuplas no importa

3.7 Tipos de tablas

- **Persistentes**. Son las tablas que crean los usuarios (normalmente los administradores). Sólo pueden ser borradas por los usuarios Se dividen en tres tipos:
    - **Bases**. Son tablas independientes. Se crean indicando su estructura y después se les añaden los datos. Cuando alguien habla de tablas, se refiere a este tipo de tablas. Las tablas base contienen tanto datos como metadatos. Por ello (debido a los datos) pueden ocupar mucho espacio en disco. Son el fundamento del modelo relacional
    - **Vistas**. Contienen una instrucción (normalmente en lenguaje SQL) la cual provoca una consulta sobre los datos de las tablas base, haciendo que se muestren los datos que cumplen las condiciones especificadas en dicha instrucción. No ocupan mucho espacio en disco (ya que no almacenan datos). Si los datos de las tablas base cambian, los de las vistas que utilizan esos datos también cambian.Las vistas permiten una visión de los datos más cómoda para los usuarios.
    - **Instantáneas o vistas materializadas**. Son vistas (es decir, consultas) que además de almacenar la instrucción SQL, almacenan una copia de los datos que muestra. De modo que la siguiente vez que usemos la instantánea, utilizará la copia de los datos (no leerá los datos originales). Las instantáneas, a diferencia de las vistas, no siempre muestran los datos actualizados. Las instantáneas actualizan los datos, refrescando el resultado cada cierto tiempo. La ventaja es que son más veloces que las vistas al no tener que acudir a los datos base; la parte mala, es que los datos que muestran pueden ser obsoletos (ya que pueden haber variado en las tablas base originales).
- **Temporales**. Son tablas que se eliminan automáticamente por el sistema. Las utiliza el SGBD como almacén intermedio de datos (resultados de consultas, por ejemplo) para acelerar la ejecución del sistema o como almacén auxiliar para instrucciones complejas.

Hay tablas temporales de tipo base, vista o instantáneas; al igual que ocurre con las persistentes.

4. 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.

> Algunos SGBDR permiten trabajar con dominios, pero su manejo es más complejo y no lo vamos a utilizar a lo largo de este curso. Lo supliremos con el tipo de dato adecuado y un conjunto de restricciones.

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.

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 (booleanos)**: 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).
- **Binario**: valor que permite guardar dentro ficheros o cadenas de bytes.
- **Fechas**: valores que permiten guardar fechas, fechas y horas, intervalos de tiempo, momentos, duraciones, ...

> Durante los próximos temas iremos trabajando poco a poco con los tipos de datos que ofrece el estándar SQL y los que tienen los SGBDR concretos que usaremos.

5. Claves

Supongamos que vamos a implementar una aplicación web, y queremos guardar en una relación a los Usuarios del sistema. En esta tabla se van a guardar los siguientes atributos: Login del jugador que será nuestro usuario, Password o Contraseña, Nombre y Apellidos, Dirección, Código Postal, Localidad, Provincia, País, Fecha de

nacimiento para comprobar que no es menor de edad, Fecha de ingreso en la web, Correo electrónico, Sexo y por último los Créditos (dinero "ficticio") que tenga.

¿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.

5.1 Superclaves

Para diferenciar una tupla de otra, 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 **superclave**.

> Una **superclave** es el atributo o conjunto de atributos de una relación (o tabla) que nos permite diferenciar una tupla (fila) de otra.

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.

Para identificar las superclaves 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 superclaves es conociendo el significado real de los atributos (campos), ya que así podremos saber si es posible que aparezcan duplicados. Es posible desechar conjuntos de atributos como superclaves 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.

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}`

Una superclave 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.

Tendríamos que elegir alguna de las superclaves para diferenciar las tuplas. ¿Cuál es la más idonea? Lo descubriremos a través de los siguientes conceptos.

5.2 Claves candidatas

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 aquellas superclaves que tengan un menor número de atributos (es decir, un menor grado).

Por tanto, cada tabla debe tener al menos una clave candidata aunque puede haber más de una.

Siguiendo con nuestro ejemplo, podríamos considerar *Login* o *Email* como claves candidatas, ya que sabemos que el *Login* debe ser único para cada usuario, a *Email* le sucede lo mismo.

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.
- **Menor grado**: no puede existir otra clave candidata que esté formada por un número menor de columnas.

Por ejemplo, si hubiéramos elegido como clave candidata `{Nombre, Apellidos y Fnacimiento}`, comprobaríamos que 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 superclave. Sin embargo, existen otros atributos, como *Login* o *email* (conjuntos de grado = 1) que también son claves candidatas, por lo que el conjunto `{Nombre, Apellidos y Fnacimiento}` sí que sería superclave pero *no sería clave candidata*.

5.3 Clave primaria

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.

De alguna manera podemos decir que ***la clave primaria es la mejor clave candidata***.

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.

¿Qué criterios seguimos a la hora de escoger la mejor clave candidata como clave primaria? No existe una regla mágica o única a la hora de hacer esta elección, pero podemos seguir una serie de pistas.

- **Tamaño menor**: ya hemos comprobado que una clave primaria debe tener el menor grado posible (menor nº de atributos). Además, nos puede interesar que el tamaño del dato a almacenar sea lo más pequeño posible. Esto permitirá tener algunas ventajas a la hora de buscar o comparar.

> Por ejemplo, un DNI tiene 9 caracteres y un email podría tener 320.

- **Que sea un atributo clave *natural***: es decir, que sea uno de los campos que tenemos que almacenar en dicha tabla.

> Si creamos una tabla de *Clientes*, y ya vamos a guardar su email, podríamos utilizarlo como clave primaria. O en el caso de un libro, podríamos usar el ISBN.

- **Que sea frecuentemente condición de búsqueda**: es decir, que sea un campo sobre el que vamos a realizar consultas con frecuencia.

> Por ejemplo, en la tabla de *Clientes* de una tintorería, suele ser habitual utilizar el teléfono como clave primaria; y también suele ser habitual que sea el dato que se utilice para buscar los datos de un cliente.

- **Es mejor añadir un campo numérico que tener una clave primaria natural compuesta**: es decir, si la relación sobre la que queremos definir la clave primaria solo tiene claves candidatas de grado 2, en ocasiones en mejor añadir un atributo *artificial* y autonumérico, y definir el mismo como clave primaria.

> Un ejemplo suelen ser las *Ventas* realizadas online. A cada venta se le asigna un número correlativo.

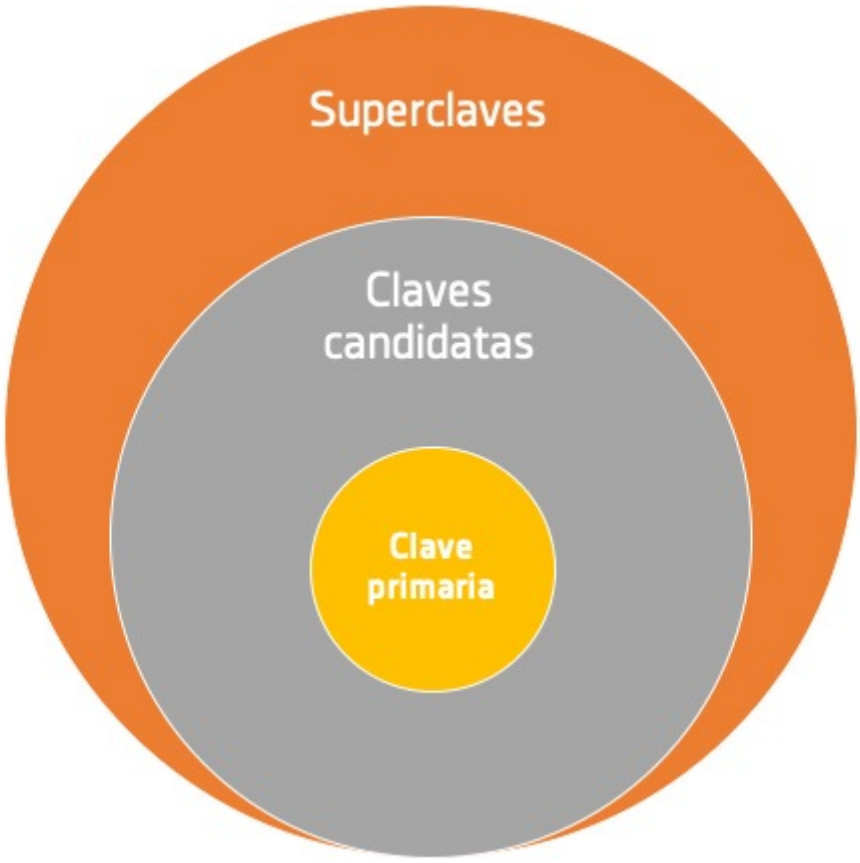
Al final, ***el mejor criterio para escoger la clave primaria será la experiencia*** del diseñador de la base de datos.

5.3 Claves alternativas

Todas aquellas claves candidatas que no hemos escogido como clave primaria se denominan **claves alternativas**.

En el ejemplo anterior de la tabla de *Usuarios*, si escogemos el atributo *login* como clave primaria, el atributo *email* sería una clave alternativa.

Este gráfico puede ilustrar sobre como unas claves están relacionadas con otras.



5.4 Claves externas, foráneas o ajenas.

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 "asociación" 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 externas representan asociaciones entre datos. Dicho de otra manera, es la existencia de la clave primaria de una tabla en 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 primaria en su tabla aparecerá en la tabla *Partidas* como clave externa. 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 externa.

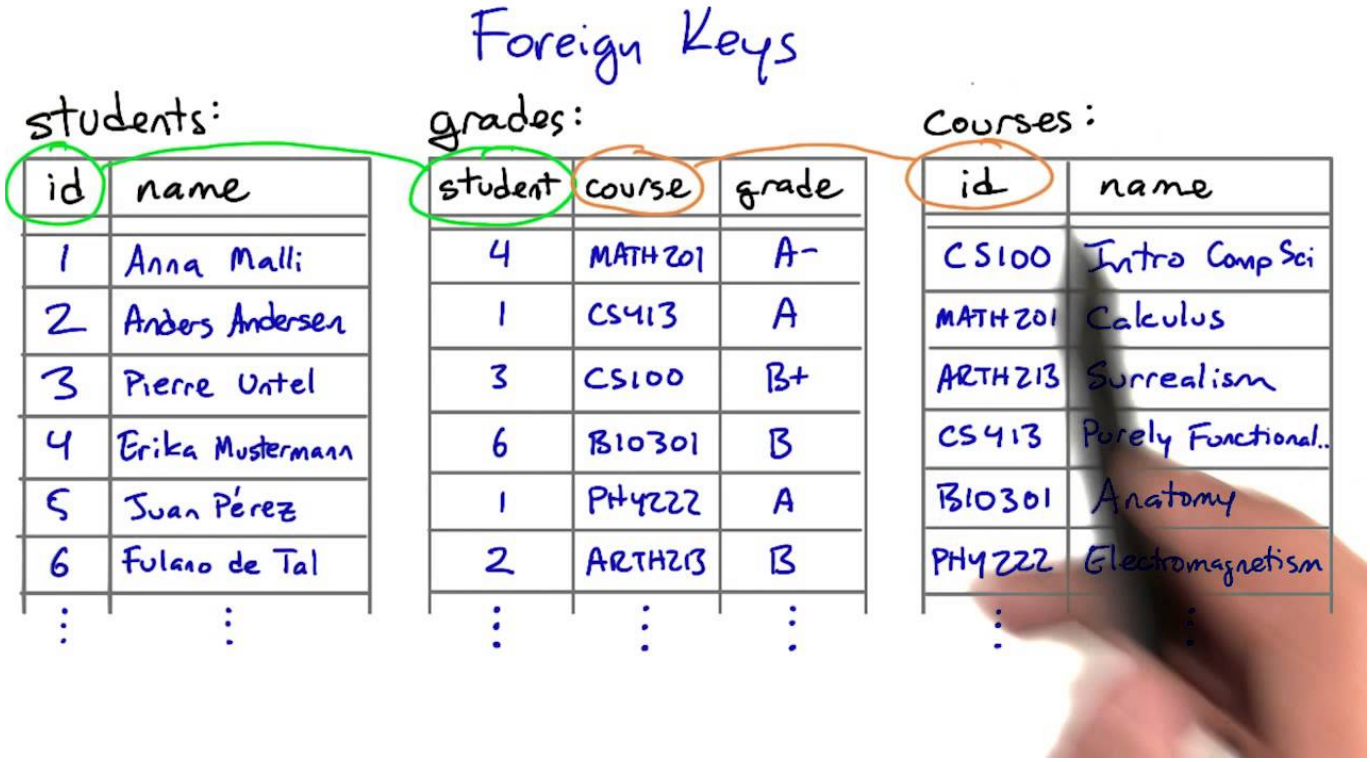
Es lógico que las claves externas no tengan las mismas propiedades y restricciones que tienen como clave primaria en su tabla, por tanto, sí que pueden repetirse en la tabla. En nuestro ejemplo, un mismo jugador puede jugar varias partidas.

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

> Esta idea, llamada **integridad referencial**, la presentaremos con mayor profundidad en un apartado posterior.

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.

En la siguiente imagen tenemos otros ejemplos de clave externa:



6. Valores NULOS

¿Qué ocurre si no conozco algún valor de un dato? ¿Si en la tabla de usuarios estoy pidiendo que se guarde el sexo y el jugador no quiere decirlo? ¿Qué puede ocurrir? Si se permite que ese dato no sea obligatorio lo que me quedaría sería un dato vacío de información.

¿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 externas 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.

> **¡OJO!** NULO no es igual que una cadena de caracteres vacía, un espacio en blanco o el valor numérico 0. Nulo es **ausencia o desconocimiento de información**.

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 tenemos los valores VERDADERO y FALSO, pero un valor NULO no es ni verdadero ni falso.

CONDICION1	CONDICION2	RESULTADOS		
x	y	x and y	x or y	not x
VERDADERO	VERDADERO	VERDADERO	VERDADERO	FALSO
VERDADERO	FALSO	FALSO	VERDADERO	
VERDADERO	NULL	NULL	VERDADERO	
FALSO	VERDADERO	FALSO	VERDADERO	VERDADERO
FALSO	FALSO	FALSO	FALSO	
FALSO	NULL	FALSO	NULL	
NULL	VERDADERO	NULL	VERDADERO	NULL
NULL	FALSO	FALSO	NULL	
NULL	NULL	NULL	NULL	

> Si quieres saber más sobre el valor NULO en las bases de datos relacionales puedes visitar el siguiente enlace: https://es.wikipedia.org/wiki/Null_%28SQL%29.

7. Restricciones

Se trata condiciones de obligado cumplimiento para que un dato forme parte de una tabla.

7.1 Inherentes

Son aquellas que no requieren que se establezcan de forma explícita, sino que son definidas por el propio hecho de que la base de datos sea relacional. Las más importantes son:

- No puede haber dos filas iguales
- El orden de las filas no es significativo
- El orden de las columnas no es significativo
- Cada atributo sólo puede tomar un valor en la intersección entre fila y columna

> Las **restricciones inherentes** coinciden con las propiedades de las tablas en el modelo relacional.

7.2 Semánticas

El modelo relacional permite incorporar restricciones personales a las tablas. Son las más importantes y son fundamentales para que la información de la base de datos sea coherente y eficiente.

> El término **semántica** hace referencia al significado de algo. En este caso, las **restricciones semánticas** son restricciones asociadas al **problema que estamos resolviendo** o para el cual estamos creando la base de datos. La base de datos de un juego online no tendrá las mismas restricciones semánticas que la de un banco.

7.2.1 Restricción de unicidad

Impide que los valores de los atributos marcados de esa forma, puedan repetirse en distintas filas. Es decir, en esa columna los valores deben ser distintos para cada fila, o bien quedar vacíos.

7.2.2 Restricción de obligatoriedad

Prohíbe que el atributo marcado de esta forma quede vacío (es decir impide que pueda contener el valor nulo, `null`) en alguna fila. También se suele indicar en las columnas que son clave alternativa.

7.2.3 Restricción de clave primaria

Marca uno o más atributos (una o más columnas) como identificadores de la tabla. En el modelo relacional, toda tabla requiere de un identificador o clave principal para asegurar que todas las filas son diferentes en cada tabla.

Esta restricción (además de marcar los datos identificativos de una tabla) prohíbe que las columnas que forman la clave primaria puedan contener valores repetidos en distintas filas o que queden vacías (nulos) en alguna fila. Es decir la restricción de clave primaria es la suma de la de unicidad y obligatoriedad.

7.2.4 Restricción de clave alternativa

Como ya se ha explicado antes, las claves candidatas que no se eligen para ser claves principales se marcan como clave alternativa. En casi todos los sistemas gestores de bases de datos no es posible marcar claves alternativas de forma explícita.

Sin embargo, si una columna es una clave alternativa, no podrá repetir valores ni quedar vacía. Es decir, las claves alternativas se marcan con restricciones de unicidad (unique) y de obligatoriedad (`not null`). Esto significa que si detectamos claves alternativas habrá que marcar estas dos restricciones sobre ellas.

7.2.5 Restricción de integridad referencial

Las claves externas son el mecanismo del modelo relacional para poder asociar datos de diferentes tablas.

La restricción de integridad referencial esta ligada a las claves externas y lo que implica es que las columnas marcadas como claves externas (*foreign keys*) no puedan contener valores que no se puedan relacionar con las clave principal de la tabla que relacionan (llamada tabla principal). Sin embargo, sí se permite dejar nulas estas columnas.

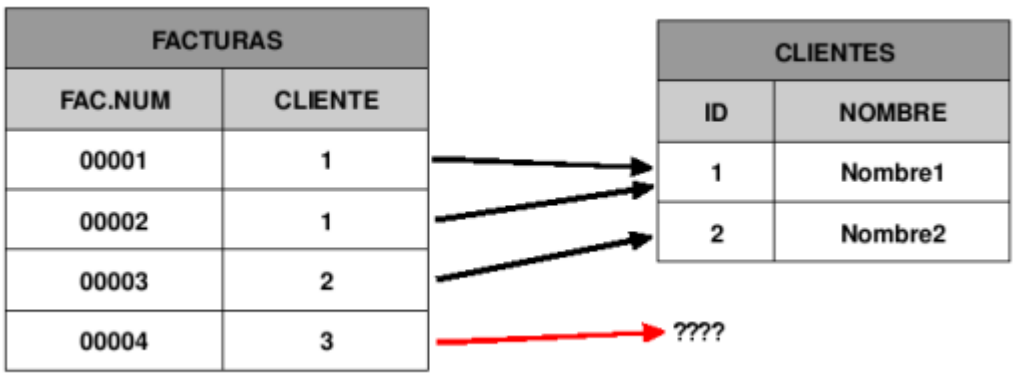


Imagen: En este caso, la fila correspondiente a la factura 00004 no es correcta, ya que el cliente con ID 3 no existe.

7.2.5.1 Políticas de actulización y de borrado.

La restricción de integridad referencial causa problemas en las operaciones de borrado y modificación de registros, ya que si se ejecutan esas operaciones sobre la tabla principal quedarán filas en la tabla secundaria con la clave externa haciendo referencia a un valor que ya no existe, y eso la propia restricción no lo permite.

Es decir no podemos cambiar la clave o eliminar clientes que tengan facturas relacionadas. Es decir si en el ejemplo anterior, borramos la fila en clientes que hace referencia al cliente con ID 1, la base de datos

reaccionará prohibiendo esta operación ya que hay dos facturas relacionadas con ese cliente. Lo mismo ocurre si intentamos cambiar el ID de dicho cliente.

Para solventar esta situación se utilizar políticas especiales cuando creemos la restricción en la clave externa.

- **Prohibir la operación** (*no action*). Esto no permitiría hacer en las claves principales ninguna operación si hay claves secundarias relacionadas con ellas. Suele ser la opción por defecto. Es muy rígida.
- **Transmitir la operación en cascada** (*cascade*). Es decir si se modifica o borra un cliente, también se modificarán o borrarán los alquileres relacionados con él.
- **Colocar nulos** (*set null*). Si modificamos o borramos un cliente, sus alquileres marcarán el antiguo código de ese cliente como nulo. Por ejemplo si borramos al cliente con ID 1, las facturas 00001 y 00002 quedarían con un código de cliente con valor `null`.
- **Usar el valor por defecto** (`default`). Se colocan un valor por defecto en las claves externas relacionadas cuando se borre o modifique la clave principal relacionada. Este valor por defecto se indica al crear la tabla (opción default).

No todas las bases de datos admiten todas estas posibles políticas a aplicar en operaciones de actualizar o eliminar. Además, podemos indicar una política al actualizar y otra al eliminar. Es decir podremos, por ejemplo, indicar nulos ante cambios en las claves principales y actuar en cascada ante eliminaciones en las claves.

7.2.6 Restricción de validación

Es una restricción que impone una condición lógica que deberá de cumplir una o más columnas cuando se la añadan o modifiquen los datos. Por ejemplo, podríamos restringir la columna llamada sueldo para que siempre acepte valores mayores de 1000; no se permitiría entonces indicar sueldos menores de 1000 en dicha columna.

A veces las reglas implican a varias columnas, como por ejemplo que la fecha de inicio sea mayor que la fecha final.

7.2.7 Disparadores

Son restricciones más complejas, presentes en sistemas avanzados de bases de datos. Se trata de código almacenado en la base de datos, cuyas instrucciones se ejecutan automáticamente cuando ocurre un determinado evento (añadir una fila, modificar filas, iniciar sesión por parte del usuario,...).

Las instrucciones pueden realizar cualquier operación permitida. Por ejemplo, podemos hacer que ningún usuario pueda añadir datos de 12 de la noche a 5 de la mañana, o que no podamos añadir una cuenta bancaria si sus dígitos de control no cumplen la compleja regla que se les aplica.

Los triggers permiten realizar restricciones muy potentes, pero son las restricciones más complejas de definir ya que implican conocimientos de programación.

8. Las 12 reglas de Codd.

Preocupado por los productos que decían ser sistemas gestores de bases de datos relacionales (RDBMS) sin serlo, Codd publica 12 reglas que debe cumplir todo Sistema Gestor de Bases de Datos para ser considerado relacional ; lo hace en el año 1985.

Las doce reglas en la práctica las cumplen pocos sistemas relacionales, pero sí los mejores. El interés actual de estas reglas, cuando el propio modelo relacional se está revisando, es que permiten detallar la manera de

funcionar de un sistema de bases de datos relacional.

Las reglas son:

1. **Información**. Toda la información de la base de datos (metadatos) debe estar almacenada explícitamente en el esquema lógico. Es decir, todos los datos se almacenan en las tablas base del sistema. Dicho de otro modo: no pueden existir datos a los que tengamos que acceder por una vía diferente a la de las tablas del modelo relacional.
2. **Acceso garantizado**. Todo dato en la base de datos es accesible sabiendo el valor de su clave principal y el nombre de la columna o atributo que contiene el dato. No hace falta saber, por ejemplo, ni el número de fila ni el de columna. Esto implica que todas las tablas tienen que tener identificador y eso nos permite acceder a una fila concreta; la columna es accesible por su nombre y no por el orden que tiene.
3. **Tratamiento sistemático de los valores nulos**. El DBMS (Sistema Gestor de bases de datos) debe permitir el tratamiento adecuado de estos valores. Esto significa que los valores nulos se manejan como una información más de la base de datos; se podrá utilizar como información en sí y por lo tanto operar con estos valores no producirá error alguno. Se permitirá realizar operaciones lógicas y de todo tipo con ellos, haciendo que el resultado sea coherente.
4. **Catálogo en línea basado en el modelo relacional**. El catálogo en línea es otro nombre para el diccionario de datos. Esta regla indica que los metadatos deben de ser accesibles usando un esquema relacional. Es decir la forma de acceder a los metadatos es la misma que la forma de acceder a los datos. Dicho de otra forma, también los metadatos se almacenan en tablas.
5. **Sublenguaje de datos completo**. Al menos, debe de existir un lenguaje que permita el manejo completo de la base de datos. Mediante este lenguaje podremos realizar cualquier operación sobre la base de datos, sea del tipo que sea.
6. **Actualización de vistas**. El SGBD debe encargarse de que las vistas muestren la última información. En ningún caso las vistas mostrarán información obsoleta. Esto implica una enorme potencia por parte del Sistema Gestor de Bases de Datos.
7. **Inserciones, modificaciones y eliminaciones de dato nivel**. Cualquier operación de modificación debe actuar sobre conjuntos de filas o registros, nunca deben actuar registro a registro. Por lo tanto los lenguajes que realizan estas tareas (DML) son de cuarta generación y no pueden ser lenguajes como C o Java por ejemplo (de tercera generación) que normalmente utilizarían recorridos fila a fila y bucles para modificar los datos, provocando un manejo menos humano de la base de datos.
8. **Independencia física**. El esquema lógico y los externos (las aplicaciones de usuario) no se deben modificar por los cambios que se realicen en la base de datos física. Es decir aunque, por ejemplo, cambiemos el nombre de un fichero de la base de datos, por ejemplo, no tendremos que modificar ni los programas de los usuarios ni siquiera la lógica (las tablas) de la base de datos.
9. **Independencia lógica**. Los cambios en la lógica de la base de datos (en las tablas), no afectan a la forma en la que el usuario accede a la base de datos. Es decir, las aplicaciones de usuario son independientes de la propia lógica. Esta independencia en la práctica no se consigue tan fácilmente como la anterior.
10. **Independencia de integridad**. Las reglas de integridad deben almacenarse en la base de datos (en el diccionario de datos), no en los programas de aplicación. Eso permite que dichas reglas sobre los datos se cumplan siempre independientemente de la forma de acceder a los mismos.
11. **Independencia de la distribución**. El sublenguaje de manipulación de datos (DML) debe permitir que sus instrucciones funcionen igualmente en una base de datos distribuida que en una que no lo es. Los programas y aplicaciones de usuarios se crean de la misma forma. Es decir las bases de datos distribuidas permiten trabajar en ellas como si fueran una base de datos normal y local.

12. **No subversión**. Si el SGBD dispone de un lenguaje de bajo nivel (normalmente será un lenguaje de tipo procedimental) para trabajar en la base de datos (como por ejemplo el PL/SQL de Oracle), este lenguaje no se puede saltar ninguna regla de las anteriores. Puede que facilite la realización de tareas este tipo de lenguaje, pero en ningún caso podrá trabajar con los datos de forma incompatible con el modelo relacional.

> En este enlace puedes encontrar un mapa conceptual con las 12 reglas de Codd.
> https://www.mindmeister.com/es/1079684487/las-12-reglas-de-codd-del-modelo-relacional?fullscreen=1

Bibliografía

1. Jorge Sánchez. Manual de Gestión de Bases de Datos. (3) Modelo relacional. https://jorgesanchez.net/manuales/gbd/modelo-relacional.html
2. José Luis Comesaña. Apuntes Desarrollo de Aplicaciones Web. Bases de Datos. Tema 2. https://github.com/statickidz/TemarioDAW/blob/master/BBDD/BD02.pdf