**Entidades & Atributos.**

**Sobre los principales objetos de manipulación dentro de una base de datos relacional.**







**Pasos para crear una base de datos relacional en nuestro proyecto (esta vez el proyecto será un sistema de "Platziblog").**

**En un proyecto real, para crear una base de datos relacional, lo primero que hay que hacer es:**

**1. Identificar las entidades (para nuestro caso, serían las entidades de un “Blog” de Platzi).**

*1ra entidad:* ***"Posts":*** *El cuerpo propiamente del Blog.*

*2da entidad:* ***"Usuarios":*** *Los que van a crear el contenido del Blog.*

*3ra entidad:* ***"Comentarios":*** *Caja de comentarios al contenido del Blog.*

*4ta entidad:* ***"Categorias":*** *Tienen que ver con las sesiones del "blog"; es decir, las pestañas que tiene el blog por separado por cada tema de interés. Las categorias pueden ser: "Política", "Deportes", "Economía", "Moda", "Tecnología", etc.*

**2. Identificar los atributos (para nuestro caso, serían los atributos de un “Blog” de Platzi).**

***Para la entidad "Posts":***

*1er atributo:* ***Un título.***

*2do atributo:* ***La fecha\_de\_publicación.***

*3er atributo:* ***el Contenido.***

*4to atributo:* ***El estatus*** *(si fue publicado o no, si está activo o no).*

*5to atributo:* ***Las etiquetas****, las cuales son Atributos Multivaluados; es decir, hay varias etiquetas realmente y se encierran en un ovalo con doble subrayado. La etiqueta es la que nos dicta que tipo de post estás leyendo,* ***son las etiquetas de las categorias****; es decir, las etiquetas que se ponen al final del post si hace parte de la sesión de "ciencias" o de la sesión de "tecnología" o de "deportes", etc. Pues bien, en un mismo post, pueden haber varias etiquetas; en ese caso, como son varias (una por post), es por lo anterior que en el atributo "etiqueta" hablamos de que sea una etiqueta "multivalor".*

*6to atributo:* ***Un id, esta sería la etiqueta "llave" o clave de nuestra entidad "Posts"****; es lo que nos dirá que todos los* ***posts*** *serán diferentes entre sí, al menos, minimamente, por el "id".*

***Para la entidad "Usuarios":***

*1er atributo:* ***Un “login”****, para poder hacer inicio de sesión con una cuenta.*

*2do atributo:* ***Un “password”****, para tener acceso a nuestra cuenta con una contraseña.*

*3er atributo:* ***Un apodo o “alias”.***

*4to atributo:* ***Un email.***

*5to atributo:* ***Un id, esta sería la etiqueta "llave" o clave de nuestra entidad "Usuarios"****; es lo que nos dirá que todos los* ***usuarios*** *serán diferentes entre sí, al menos, minimamente, por el "id".*

***Sobre las relaciones***

**Las relaciones es la manera en la que empezamos a ligar nuestras diferentes entidades u objetos (De aquí surge la importancia de los Diagramas entidad-relación)**

Las relaciones se representan con un **Rombo.**

**Cómo funcionan las relaciones?**

Supongamos que tenemos la entidad *Automovil* y la entidad *Dueño*, siendo en total dos (2) entidades:

Automovil

Dueño

Ahora, lo que nos permite vincular o relacionar ambas entidades son las ***relaciones,*** *éstas por convención se definen por medio de* ***verbos.*** En este caso, un verbo más apropiado de la relación que hay entre las entidades “**automovil” y “dueño” *es “*tener”.** Quedando visualmente este vinculo de entidades y relaciones así:

Automoviles

tienenee

Dueños

Es necesario el uso de *líneas conectoras* para poder ligar o hacer la unión entre las entidades, a esta representación visual se le llama: **“Diagramas entidad-relación”**.

***Otro ejemplo de relación entre dos entidades podría ser:***

Jugadores

pertenecen

Equipos

Lo que distingue una entidad de un atributo es que el atributo es algo propio a una entidad; es decir, es una caracteristica de la entidad u objeto; mas, cuando se habla de dos entidades, se habla de dos objetos por separado y completamente independientes entre sí. **Sin embargo, hay un tipo de atributo en especifico que recibe un tratamiento igual al de una entidad en términos de *relaciones, y son: Los atributos Multivaludos.*** Veamos esto en un caso que ya hemos trabajado.

Laptops

tienen

discos\_duros

Si se da cuenta, realmente los *discos duros* no son independientes a una Laptop; antes, son un elemento o caracteristica propia de ellas (de las laptops); es decir, los *discos duros* no son una entidad como tal; sin embargo, como es un *atributo multivaluado*, recibe el mismo tratamiento en términos de relaciones que cualquier otra entidad (convertiendose, en ese sentido, en una entidad más). Lo anterior se debe a que los atributos multivaluados son más complejos y pueden relacionarse de varias formas con su entidad.

Ahora, para definir o saber puntualmente la cantidad de, por ejemplo, los *discos duros* que tiene una *laptop* se hace por medio de una nueva propiedad llamada: ***Cardinalidad;*** es decir, la ***cardinalidad*** vendría siendo lo que nos permite saber la cantidad de entidades o de atributos multivaluados que hay de lado y lado en cada relación resuelta. O también se puede definir la ***Cardinalidad*** como: **La facultad de saber la cantidad numérica de valores que tiene un atributo multivaluado (o una entidad -otra) para una entidad en cuestión (al ser relacionadas ambas)”.**

***Cardinalidad: casos o diferentes tipos.***

***Cardinalidad: 1 a 1***



Cuando se habla de ***cardinalidad: 1 a 1,*** nos referimos a que sólo hay un valor, para lado y lado, en la relación etiqueta-etiqueta o etiqueta-atributo multivaluado; en nuestro ejemplo, para la entidad “persona”, nos estaríamos refiriendo a una sola persona como tal por separado; por otra parte, para el atributo multivaluado “datos-contacto”, nos referimos a los datos de contacto correspondientes a una única persona precisamente. ***Esta representación gráfica reflejada es un diagrama entidad-relación.*** Sin embargo, existe otro tipo de diagrama para representar visualmente la misma información dada por un caso de cardinalidad: 1 a 1,... y son los ***Conectores*** *de un diagrama fisico de una base de datos****.***



El primer conector básicamente nos dice lo mismo: *“de un lado tengo uno y del otro lado tengo otro”;* mas, el segundo conector también significa lo mismo, sólo que refuerza la declaración de que en un lado efectivamente tengo uno *(y sólo uno)* y del otro lado tengo uno también *(y sólo uno).* Esa expresión de *“sólo uno”* se simboliza gráficamente con la doble raya a los extremos del conector.

***Cardinalidad: 0 a 1 (o Cardinalidad: 1 a 1 opcional)***

Acá puede haber la opción de que no exista ninguno de uno de los dos lados. Veamos el siguiente caso.



Si analizamos la “sesión\_actual”, la sesión de un usuario, con el “usuario” mismo; podemos decir que toda “sesión\_actual” tiene que tener un usuario, pero un usuario puede *no estar en sesión* en determinado momento; por lo tanto, la *cardinalidad se define como de: 0 a 1,* pues, puede que no haya una *sesión actual.*

***Cardinalidad: 1 a N (o 1 a muchos)***

***La Cardinalidad: 1 a muchos*** quiere decir, como su nombre indica, que de un lado tenemos uno pero del otro lado tenemos muchos. Veamos el siguiente ejemplo.



Como logra observar, se plantea que una persona puede tener muchos automoviles.

***Cardinalidad: 0 a N (o 0 a muchos)***

Acá puede haber la opción de que no exista ninguno de uno de los dos lados, pero muchos en el otro lado restante. Veamos el siguiente caso.



Si analizamos al “paciente”, el que ocuparía la habitación de hospital, con la “hab\_hospital” misma; podemos decir que todo “paciente” tiene que tener asignado una “hab\_hospital”; pero, en algún caso, muchas de esas *habitaciones de hospital* quedan completamente desocupadas de *pacientes;* por lo tanto, la *cardinalidad se define como de: 0 a N,* pues, puede que no haya ningún *paciente* para una cantidad considerable de *habitaciones de hospital* disponibles.

***Cardinalidad: N a N (muchos a muchos)***

Esta es, quizás, el tipo de *cardinalidad* más especial; se separa un poco del resto de *cardinalidades* porque no contempla a una “*entidad principal”* con un único valor posible. *Generalmente son muchos de lado y lado*. Veamos el siguiente ejemplo.



Un alumno puede estar inscrito o tomar varias clases; puede, por ejemplo, tomar Español, Matematicas o Fundamentos de bases de datos (1-N); pero adicionalmente una clase, cualquiera de esas tres mencionadas, puede estar *relacionada* o contiene a varios alumnos (N-1). El último conector de esta cardinalidad es más extricto que el primero; pues, nos cuenta que, en todos los casos, siempre habrá *muchos* de un lado y *muchos* de otro: no puede haber *0s,* no pueden haber *1s*, siempre habrán *muchos.* Ahora, lo más común es que se dé el primer conector y no el último entre la *cardinalidad: N a N.*

***Diagramas entidad-relación: a detalle***

**Es preciso decir que la introducción al tema de *Entidades, atributos y, posteriormente, relaciones* es fundamental para profundizar sobre el tema que nos concierne ahora: *Los diagramas entidad-relación.***

***Qué es un diagrama ER?***

Un diagrama lo que nos va a hacer es como una especie de mapa. Pongamos así. El equivalente a, por ejemplo, los planos que diseñan los arquitectos antes de construir un edificio, para nuestro caso, serían los diagramas ER antes de construir nuestras bases de datos relacionales. Ya con el diagrama es que se entenderá bien cuáles son las entidades con las que vamos a trabajar, sus relaciones y cuáles son los atributos de dichas entidades. Dicho eso, entonces, sabríamos qué papel jugarían todos esos elementos dentro de los sistemas o aplicaciones que vamos a desarrollar con nuestras bases de datos. Pongamos todo esto en contexto con nuestro mismo proyecto: *PlatziBlog.*



Si se da cuenta, hemos agregado al conjunto de entidades a las “etiquetas”; pues, si se recuerda, habíamos comentado que muchas veces los atributos multivaluados recibían un tratamiento similar, o se convertirían, en entidades separadas también. Lo anterior tiene sentido si recordamos que los atributos multivaluados tienen su vida propia; para este caso puntual, *están en más de una noticia, en más de un tipo de Blogposts.* Pasa que las *etiquetas* ya no dependen directamente de la entidad *Post*; sino que, pueden estar asociadas a diferentes entidades y no sólo a los *Posts.* Entonces, éstas serían nuestras 5 entidades para nuestra base de datos relacional del proyecto “Platziblog”.

***Ahora, vamos a empezar a observar nuestras relaciones...***

**Relación 1:** Un usuario puede *escribir* muchos Posts.

**Relación 2:** Un usuario puede *escribir* muchos comentarios.

**Relación 3**: Un post puede *tener* muchos comentarios.

**Relación 4:** Una categoria (un tema central) puede *tener* muchos posts; por ejemplo, la sesión de *Economía* puede tener varios posts que escriban de ella, que escriban sobre varios temas relacionados a lo *económico.*

**Relación 5:** Una etiqueta (una etiqueta de un tema central) puede tener muchos posts; pero, también, un post puede tener muchas etiquetas: creando así una cardinalidad *muchos a muchos.*



***Diagrama Físico***

**El diagrama físico deriva del anterior, *diagrama entidad-relación,* es un complemento al mismo que viene a ultimar los detalles necesarios para dar paso a la creación, ahora sí, de una base de datos relacional.**

***Tipos de dato: los básicos en cualquier base de datos (1er elemento del diagrama físico)***



***Aclaremos diferencias.***

**Tipos de datos de tipo *texto***

En los tipos de dato de tipo ***texto,***la diferencia que hay entre CHAR(n), que te permite almacenar caracteres y cadenas de texto; y VARCHAR(n), que también sirve para almacenar cadenas,.. consiste en la optimización del uso de memoria; es decir, *de cuándo saber hacer lo mismo con menos.*

Si conoces a priori la cantidad de caracteres que vas a necesitar o a usar en una cadena de texto; dicho de otra manera, que serán asignados con la declaración y no solicitados por teclado, lo recomendable es que no ocupes más espacio de memoria que el que necesitas extrictamente para almacenarlos; y, para ese caso, lo ideal sería usar un tipo de dato CHAR(n); sin embargo, por ejemplo, en los casos en los que se solicita por teclado que un usuario independiente a uno sea quien *meta* los valores (caracteres), para una cadena de texto en cuestión, no es una buena práctica declarar un tipo de dato CHAR(n) porque no sabríamos con anticipación la cantidad exacta de memoria que se va a ocupar para ella; por lo que podríamos limitarla o, en el caso contrario, darle una cantidad de memoria excesiva que bien podría haber sido destinada a otras tareas u otras variables; dicho lo anterior, lo ideal sería usar, para estos casos, el tipo de dato VARCHAR(n) que es más flexible, más dinámico y no establece a priori un límite de espacio de memoria para asignar a una variable.

Ahora, el tipo de dato TEXT sirve para almacenar cadenas de texto también, pero que resultan ser muy grandes. VARCHAR(n), en general, tiene un límite de 255 caracteres; cuando se necesite guardar más caracteres que eso, se usa siempre el tipo de dato TEXT; en nuestro caso, por ejemplo, será el tipo de dato que usaremos para guardar el contenido de nuestros *Posts.*

**Tipos de datos de tipo *numérico***

Estos tipos de datos nos ayudan a almacenar números y a hacer operaciones matematicas, hay de varios tipos: INTEGER, que nos permite almacenar números enteros. Los siguientes tipos de datos son subtipos de INTEGER: BIGINT, que almacena enteros muy grandes, superiores a 99; y SMALLINT, que almacena enteros más pequeños, que son menores a 99; todo esto es ideal, precisamente, para lograr una mayor eficiencia y optimización en el uso de memoria temporal y, en consecuencia, para desarrollar una base de datos más *rapida.*

En cuanto al tipo de datos DECIMAL(n, s) o NUMERIC(n, s), éstos reciben dos parametros como datos de entrada.En *n* se añade el número entero y en *s* se añaden los números decimales.

**Tipos de datos de tipo *fecha/hora***

Estos tipos de datos son muy útiles, por ejemplo, en nuestro caso actual (sistema Platziblog), es con este tipo de datos que se podrá insertar una fecha de publicación de los *posts* que publiquemos. Adicionalmente, con estos tipos de datos, al interior del sistema, nos sirve para saber cuándo fue creado un registro, cuando alguien lo modificó, cuando alguien lo borró, por ejemplo.

DATE, nos permite contener la fecha a secas (año, mes, día).

TIME, nos permite contener la hora del día de las 24 horas.

Por último, DATETIME y/o TIMESTAMP, lo que hacen es justamente registrar ambos: fecha y horas (incluso con registros a milisegundos). Es usado para datos de *tiempo* más precisos.

**Tipo de datos de tipo *lógico***

BOOLEAN, significa que puede tener básicamente dos valores (binario): verdadero (1) o falso (0), nos sirve de manera aleatoria para hacer una validación entre una bandera; por ejemplo, decir si un Blogpost está activo o inactivo; de esta manera, en consecuencia, cuando queramos sacar sólo los Blogposts que se encuentren activos, podemos utilizar un *booleano* para dejar por fuera todos los demás.

***Constraints: restricciones (2do elemento del diagrama físico).***

**Las restricciones son las reglas que le pasamos a las bases de datos, sus limitaciones; por ejemplo, *qué tipo de datos podemos meter, cuantos tipos de datos en sí, etc...***

***Hay de varios tipos.***



NOT NULL: Cuando tú metes un registro o una serie de datos a una tabla de una base de datos, el valor por defecto siempre es el valor nulo o *null.* Lo que estás diciendo con esta regla es: en esta columna particular, por ejemplo, en la columna del *nombre,...* no se va a permitir que vaya en blanco; por lo tanto, agregaría la restricción NOT NULL a mi tabla, más precisamente, en mi columna *nombre.* Quiere decir que si te tratan de mandar un registro en el que no viene el *nombre* de la persona, la base de datos inmediatamente va regresar un error exigiendo que se complete el campo del *nombre* y, hasta que no se complete, dicho registro no se podrá guardar en la base de datos. Es muy interesante el uso de este recurso porque te ayuda a definir los datos obligatorios que esperas recibir sí o sí dentro de tu sistema.

UNIQUE: Básicamente es decir que un valor pasado a un campo, de una columna seleccionada, debe ser *único* en toda nuestra tabla. Un ejemplo muy claro que ya hemos visto anteriormente es el registro de un *email* por parte de los usuarios.En una tabla de *usuarios* queremos que nadie tenga el mismo *email* dos veces; si yo tengo mi usuario, el cual saqué con una cuenta de mi correo electronico, no es posible que otra persona saque otra cuenta usando el mismo correo o *email* que yo usé; explicado lo anterior, esta limitación es algo que como regla deberiamos establecer,... entonces, para esos casos especiales, nos conviene usar la restricción: UNIQUE. Entonces, en resumen, si establecemos la restricción UNIQUE en la columna *email,* cuando traté alguien más sacar una cuenta con mi correo electronico o con un correo de un tercero ya en uso, va a regresar un error diciendo: “No podemos meter este registro porque ya existe otro con el mismo correo electronico”.

PRIMARY KEY: Anteriormente en los diagramas hemos visto que tenemos la necesidad de identificar de manera única (y obligatoriamente) los registros que existen en una tabla, eso lo hacemos a través de un *campo clave o de un campo llave*; el campo clave, justamente, va a tener esta restriccón PRIMARY KEY; la cual, nos da una serie de ventajas. En primer lugar nos garantiza que es NOT NULL y que es UNIQUE (lo que necesitamos para esos casos), esto porque si tu estás metiendo un campo, que es llave, y que nos ayude a identificar de manera única un registro en una tabla; quiere decir que si no los metemos, nos va a empezar a devolver errores porque ya va a haber valores nulos; y, también, si los metemos repetidos, pues entonces, ya no se identifica de manera única. Además de lo anterior, esta restricción, nos va a permitir hacer la unión entre una tabla y otra, o entre una entidad y otra. Es la PRIMARY KEY la que nos ayuda realmente hacer las relaciones entre entidades. Todo campo declarado con PRIMARY KEY es un índice o index (ya explicaremos qué es esto).

FOREIGN KEY: Se traduce como *llave foranea.* Por ejemplo, cuando queremos juntar dos tablas y decir que “esta tabla” esta relacionada con “esta otra”, lo que hacemos es que la *primary key* de una tabla se añade como *foreign key* en la otra. De la anterior manera es que se liga una tabla con otra. Esta es la función de una *foreign key,* es una llave foranea que viene de otra tabla, viene de afuera. La *foreign key* debe tener las mismas caracteristicas que la *primary key*; es decir, que sean del mismo tipo; pero, en este caso, no tienen que ser *únicas* porque nuestra tabla foranea sí se puede repetir.

CHECK: Nos permite definir una regla, la que sea que queramos, que hace un simil con el mundo real; y que nos permite, justamente, hacer reglas de negocio y reglas que obedezcan a lo que nuestro cliente o nuestro caso de uso (proceso) nos está pidiendo. Es como un condicional.

DEFAULT: Cuando tú no quieres que el tipo de dato por defecto sea NULL, al no meter un dato dentro de un campo o registro; puedes definir, por medio de DEFAULT, que sea de otro valor. Sirve especialmente en los casos en que, por ejemplo, digamos, hay registros de inventario que llenar y como bien sabe se llenan con valores numéricos; entonces, podemos hacer que nuestro sistema sobre-entienda que, ante un campo o registro vacio, esto se interprete como que no hay existencia para una determinada mercancia; entonces, en ese caso, es más conveniente que el valor por default sea 0 en vez de NULL.

INDEX: Lo que nos va a permitir es hacer busquedas más rapidas en nuestra tabla de base de datos, esto es muy útil cuando tenemos muchos registros; ahora, el índice tiene una desventaja y es que cuando existe un índice en una columna, cada que añadimos registros, se vuelve más lento porque cada que añades un nuevo registro tienen que volver a indexar (o indizar) toda la info. de la base de datos.