Bases de Datos

Última actualización: *Feb 26 2023*

Temas

Introducción

El mundo moderno está completamente centrado en la información y los datos.

¿Cuál es la diferencia entre información y datos?

Los **datos** son una unidad singular de conocimiento. No tiene valor intrínseco por sí mismo. No podemos extraerle significado sin saber más al respecto.

La **información** es algo que podemos vincular a los datos, para poder atribuirles un significado.

Por ejemplo: El número 38 es un dato. Saber que 38 es la edad de Jon es información.

Una **base de datos** es una colección de información cuidadosamente organizada en un sistema.

La tecnología que permite organizar los datos y representar la información esencial para un sistema de información se denomina **Sistema Gestor de Base de Datos** (***SGBD***) o por sus siglas en inglés (***DBMS***) ***Data Base Management System***.

Un ***DBMS*** es un software que encapsula los datos de una base de datos y nos proporciona una forma de almacenarlos, recuperarlos, editarlos, conservarlos y mucho más.

Le pedimos a un *DBMS* que sea eficiente, que almacene datos de forma privada y segura, que maneje grandes cantidades de datos.

Aunque una base de datos puede ser tan simple como un archivo de texto separado por comas (*CSV*), para aplicaciones robustas se recomienda utilizar *DBMS* como *MySQL*, *PostgreSQL*, *MongoDB*, *Oracle*, etc.

Algunas ventajas de los *DBMS* son:

- Están optimizados para almacenar y manipular grandes cantidades de datos.
- Ofrecen mayor seguridad y administración de la información.
- Permiten a varios usuarios acceder y manipular la información de forma concurrente.
- Garantizan la integridad de los datos.

Un *DBMS* puede contener muchas bases de datos.

Los *DBMS* se pueden dividir en dos grandes grupos: relacionales (***SQL***) y no relacionales (***NoSQL***).

🔼 Regresar

Tipos de Bases de Datos

Cuando desarrollamos una aplicación y nos encontramos en el momento de elegir una base de datos, siempre llegamos a un punto en el que nos hacemos la misma pregunta:

> ¿Qué tipo de base de datos debo elegir?

Independientemente de los distintos gestores de bases de datos que existen en la actualidad, esta pregunta siempre suscita la duda de sí debemos elegir una base de datos **relacional** o **no relacional**.

Para responder esta pregunta, es muy importante conocer las diferencias entre una y otra y, sobre todo, entender el tipo de aplicación que estamos realizando, ya que una mala elección de nuestra base de datos puede dar lugar a una larga lista de problemas durante el desarrollo de la aplicación.

Bases de Datos Relacionales

Se caracterizan por ser una colección ordenada de registros que se organizan en un conjunto de tablas. Una tabla es muy parecido a una hoja de cálculo, y los registros se organizan en filas y columnas. Estas tablas se **relacionan** entre sí. Para acceder a los datos, se usa el Lenguaje de Consultas Estructuradas, mejor conocido por sus siglas en inglés como ***SQL***, ***Structured Query Language***.

Con *SQL* podemos obtener y alterar datos de una forma organizada siempre y cuando tengamos en cuenta cuál es la estructura de la base de datos con la que estamos trabajando. Para ello, utilizaremos los distintos comandos que *SQL* pone a nuestra disposición.

Las tablas de una bases de datos relacional se organizan a través de identificadores. De este modo, cada tabla tiene un identificador único que es el que va a establecer su relación con el resto de tablas. A su vez, estos identificadores hacen que sea más fácil organizar cada una de las tablas por separado.

Los *DBMS* relacionales se caracterizan por modelar la información en tablas que se relacionan entre sí. Ejemplos son *MySQL*, *MariaDB*, *PostgreSQL*, *SQLServer*, *Oracle*, entre otros.

Bases de Datos No Relacionales

Las bases de datos no relacionales están diseñadas para modelos de datos con estructuras más específicas y que no necesitan ser relacionados con otros. Cada entidad funciona de forma independiente y son mucho más sencillas que las relacionales.

Esta sencillez de acceso y ordenación hace que en el panorama actual del *Big Data* estén cobrando mucha relevancia, ya que al no tener una estructura definida como en las relacionales, se puedes tener redundancia de datos, es decir, podemos tener datos repetidos.

¿Por qué se permite esto? Porque lo que buscamos es mejorar el rendimiento y se prioriza el acceso rápido por sobre la normalización e integridad de los datos. En este tipo de modelos se requiere ahorrar poder computo para poder procesar la mayor cantidad de datos en el menor tiempo posible.

Las bases de datos no relacionales pueden tener identificador único, pero este no se usará para relacionar unos registros con otros. La información se organiza normalmente mediante documentos y es muy útil cuando no tenemos un esquema exacto de lo que se va a almacenar.

Con respecto a los formatos que se utilizan en las bases de datos no relacionales, podríamos decir que el más popular es el de un **documento** que es un objeto con una clave y un valor para que el acceso a la información se pueda realizar de forma sencilla.

A todo este conjunto de bases de datos no relacionales, también se les asocia con el concepto de ***NoSQL*** que significa: ***Not Only SQL***, para hacer referencia a que no sólo la información de un sistema debe almacenarse en modelos relacionales.

Los *DBMS* no relacionales modelan la información de diferentes formas: documentos, llave-valor, grafos, entre otras. Ejemplos de *DBMS* no relacionales son *MongoDB*, *Redis*, *Apache Cassandra*, *Firebase* entre otras.

¿Cuándo utilizar *SQL* o *NoSQL*?

Después de esta explicación, podemos decir que ambos tipos de bases de datos son útiles y dependerá del tipo de aplicación que queramos realizar la elección de una u otra base de datos.

Así, si queremos desarrollar una aplicación de tipo contable, de inventario o de información de clientes, es probable que el modelo relacional se adapte mejor a nuestras necesidades. En este tipo de aplicaciones, normalmente habrá más de una tabla que tenga relación con el resto, por lo que una base de datos relacional será más útil y podrá representar mejor nuestra aplicación.

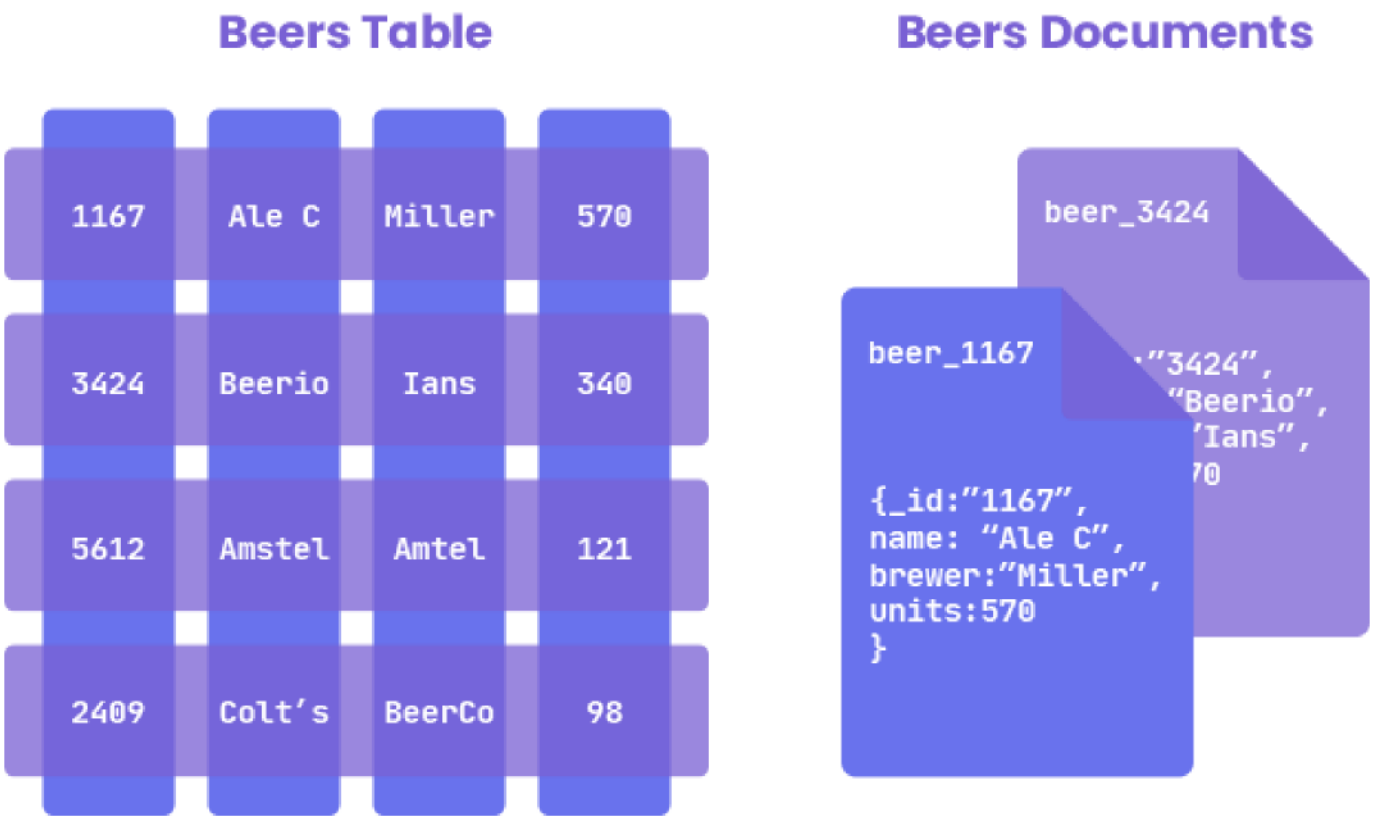
Si por el contrario nuestra aplicación necesita de un sistema en el que los datos que se vayan a almacenar no necesitan relacionarse unos con otros, y además no tenemos certeza de que todos los datos tengan la misma estructura, entonces usaremos una base de datos no relacional. Por ejemplo puede ser una base de datos en la que solo queramos almacenar las estadísticas de comportamiento de un usuario al visitar un sitio, o una base para recolectar sus datos biométricos, esta información puede cambiar de una persona a otra, dependiendo de sus dinámicas con el sitio o de su estado de salud y condición física para los biométricos. Otro ejemplo sería la creación de una galería de fotos en *Facebook* o *Instagram*, cada usuario puede subir la cantidad de fotos que guste sin que exista una estructura determinada. Las estadísticas de progreso de un videojuego también serían un buen ejemplo de modelo no relacional, ya que la información de un jugador a otro puede variar.

Todos estos ejemplos almacenan la información y no la relacionan entre sí.

Por todo lo anterior, la elección del tipo de base de datos es algo muy importante y que no hay que tomarlo a la ligera. La elección de un *DBMS* erróneo puede traer consecuencias fatales en el desarrollo de un proyecto, por lo que es muy importante dedicarle tiempo y ver qué tipo se adapta mejor con el modelo de nuestro proyecto.

La principal diferencia entre las bases de datos *SQL* y *NoSQL* es su estructura de almacenamiento de datos.

Las bases de datos *SQL* utilizan una estructura organizada y relacional, mientras que las bases de datos *NoSQL* utilizan una estructura más flexible y escalable. Ambas tienen sus propias fortalezas y debilidades, y la elección de una depende de las necesidades específicas de la aplicación y el tipo de datos que deben almacenarse.



Aquí hay algunas consideraciones que puedes tener en cuenta al elegir entre un modelo de base de datos *SQL* y *NoSQL*:

SQL

- **Relaciones estructuradas**: Si tu aplicación depende de relaciones estructuradas y complejas entre tus datos, es posible que desees considerar una base de datos *SQL*. Ya que permiten modelar relaciones complejas entre sus tablas y utilizar lenguajes de consulta estandarizados para realizar consultas complejas.
- **Datos estructurados**: Si tus datos tienen una estructura bien definida y poco probable que cambie con frecuencia, una base de datos *SQL* es una buena opción, ya que permiten definir esquemas rigurosos que garantizan la integridad de los datos y evitan la introducción de datos erróneos o inconsistentes.

NoSQL

- **Escalabilidad**: Si tu aplicación está destinada a crecer rápidamente y manejar una gran cantidad de datos, es posible que desees considerar una base de datos *NoSQL*. Estas son más escalables que las bases de datos *SQL* y se adaptan mejor a grandes cantidades de datos no estructurados.
- **Datos no estructurados**: Si tus datos no tienen una estructura bien definida o cambian con frecuencia, es posible que desees considerar una base de datos *NoSQL*. Estas son más flexibles que las bases de

datos *SQL* y te permiten modelar tus datos de manera más natural.
- **Aplicaciones en tiempo real**: Si tu aplicación requiere una respuesta rápida y en tiempo real, es posible que desees considerar una base de datos *NoSQL*. Estas suelen ser más eficientes que las bases de datos *SQL* en términos de rendimiento y escalabilidad en aplicaciones en tiempo real.

En última instancia, la elección entre un modelo de base de datos *SQL* o *NoSQL* depende de las necesidades específicas de tu aplicación y de la naturaleza de tus datos. Es importante evaluar cuidadosamente tus requisitos y considerar las fortalezas y debilidades de cada opción antes de tomar una decisión.

🔼 Regresar

Entidades y Atributos

Una **entidad** es un objeto del mundo real que se pretende controlar dentro del sistema, por ejemplo: una persona, un producto, una cuenta, un servicio, una empresa, una compra, etc.

Las entidades al ser objetos, van a tener características que las describen, a estas propiedades se les llama **atributos** de la entidad.

Por ejemplo la entidad persona tiene atributos tales como: nombre, apellidos, fecha de nacimiento, domicilio, teléfono, correo, etc.

Lo primero que tenemos que hacer al diseñar una base de datos es hacer un listado de las entidades que se verán involucradas en el sistema y de sus atributos.

Tipos de Entidades

De Datos

Las entidades de datos son aquellas que alimentan el sistema de información. En ellas se almacenan y se interactúa con los datos.

Pivotes

Las entidades pivotes son las que **relacionan** la información de 2 o más entidades. Nos ayudan a mantener consistencia e integridad en el sistema y evitan la duplicidad de datos. También suelen llamarse entidades de enlace o asociación.

Por ejemplo en el proceso de una venta, una entidad pivote puede almacenar la relación de qué y cuántos productos se adquirieron en dicha venta, además de relacionar dichos productos con la información del cliente que los compró.

Catálogos

Los catálogos son entidades que sus registros son una lista o relación ordenada con algún criterio y por tal motivo su información debe estar precargada en el sistema, antes de comenzar a introducir información en

el.

Una lista de códigos postales, colonias, municipios, cuidades o países son un buen ejemplo de entidades cátalogo.

Códigos Postales

- Códigos Postales México
- CPs México en formato TXT
- Consulta de CPs México

Países

- Información Paises

🔼 Regresar

Tipos de Datos

En las bases de datos existen varios tipos de datos que se pueden almacenar y manejar. Algunos de los tipos de datos más comunes incluyen:

- **Números enteros**: se utilizan para almacenar números enteros, como por ejemplo, edad, cantidad de productos, etc.
- **Números de punto flotante**: se utilizan para almacenar números con decimales, como por ejemplo, precios, coordenadas, etc.
- **Cadenas de texto**: se utilizan para almacenar caracteres y texto, como por ejemplo, nombres, direcciones, descripciones, etc.
- **Fechas y horas**: se utilizan para almacenar fechas y horas, como por ejemplo, fechas de nacimiento, fechas de entrega, etc.
- **Booleanos**: se utilizan para almacenar valores verdaderos o falsos, como por ejemplo, si un usuario está activo o no.
- ***Blobs* y archivos**: se utilizan para almacenar archivos grandes como imágenes, videos, audio, etc.
- **Datos geográficos**: se utilizan para almacenar información geográfica, como por ejemplo, ubicación, direcciones, etc.

Estos son solo algunos ejemplos de los tipos de datos que se pueden almacenar y manejar en una base de datos. El tipo de datos que utilices depende de las necesidades específicas de la aplicación y de la naturaleza de los datos.

🔼 Regresar

CRUD

CRUD es un acrónimo que significa ***Create, Read Update & Delete***, es decir: "Crear, Leer, Actualizar y Eliminar".

Se refiere a las 4 operaciones básicas que se pueden realizar en una base de datos, es decir, la capacidad de crear nuevos registros, leer, actualizar y eliminar los registros existentes.

Estas operaciones se consideran la funcionalidad básica que se espera de cualquier sistema de gestión de bases de datos, y suelen estar implementadas de manera nativa en la mayoría de los *SGBD*.

Estas operaciones se utilizan tanto en la administración de objetos y privilegios de la base de datos como en la gestión de los datos mismos.

🔼 Regresar

Lógica de Negocio

La lógica de negocios es el conjunto de reglas, políticas y procesos que describen cómo se lleva a cabo el negocio.

En el modelado de una base de datos, la lógica de negocios se refiere a la representación de las reglas y procesos de negocios en el modelo de datos.

Estas reglas y procesos incluyen cosas como la validación de los datos, la validación de las restricciones de negocios, la definición de las relaciones entre las entidades, y la definición de cómo se deben calcular ciertos valores.

La incorporación de la lógica de negocios en el modelo de datos es importante porque permite asegurarse de que los datos estén correctamente validados y se respeten las restricciones de negocios antes de ser almacenados en la base de datos.

También permite a los desarrolladores entender mejor cómo los datos se relacionan y se utilizan en un sistema, lo que puede ser útil a la hora de realizar tareas de mantenimiento o mejora.

Además, la lógica de negocios puede ser reutilizada en diferentes partes de la aplicación, lo que reduce el esfuerzo y el tiempo necesarios para implementar la misma lógica en múltiples lugares.

🔼 Regresar

Llaves

Una llave en bases de datos es un identificador que permite hacer único a un registro de información tenemos 2 tipos: primarias y foráneas.

Llave Primaria

Identifica un registro como único dentro de la entidad a la que pertenece. En nuestro listado de atributos pondremos las siglas ***PK*** de ***Primary Key*** delante del atributo que sea llave principal.

Llave Foránea

Relaciona los datos de un registro de una entidad con los de otra, o con un registro distinto de la misma entidad. En nuestro listado de atributos pondremos las siglas ***FK*** de ***Foreign Key*** delante del atributo que sea llave foránea.

Atributos Únicos

En algunas ocasiones vamos a necesitar que algunos atributos de la entidad sean **únicos**, es decir que no existan datos duplicados en el atributo, sin que necesariamente sea una llave primaria o foránea.

Esta característica se utiliza a menudo para asegurarse de que los datos sean consistentes y no haya duplicados en la entidad. Por ejemplo para que un usuario no pueda crear 2 cuentas diferentes con un mismo correo o número de teléfono.

Datos que suelen definirse como atributos únicos podrían ser el *DNI*, *email*, teléfono móvil, nombre de usuario o alias, número de placas de un automóvil, etc.

🔼 Regresar

Relaciones

Las relaciones son asociaciones entre entidades que se crean para recuperar y vincular datos.

Para crear una relación semánticamente utiliza un **verbo** para relacionar las entidades en cuestión.

Tipos de Relaciones

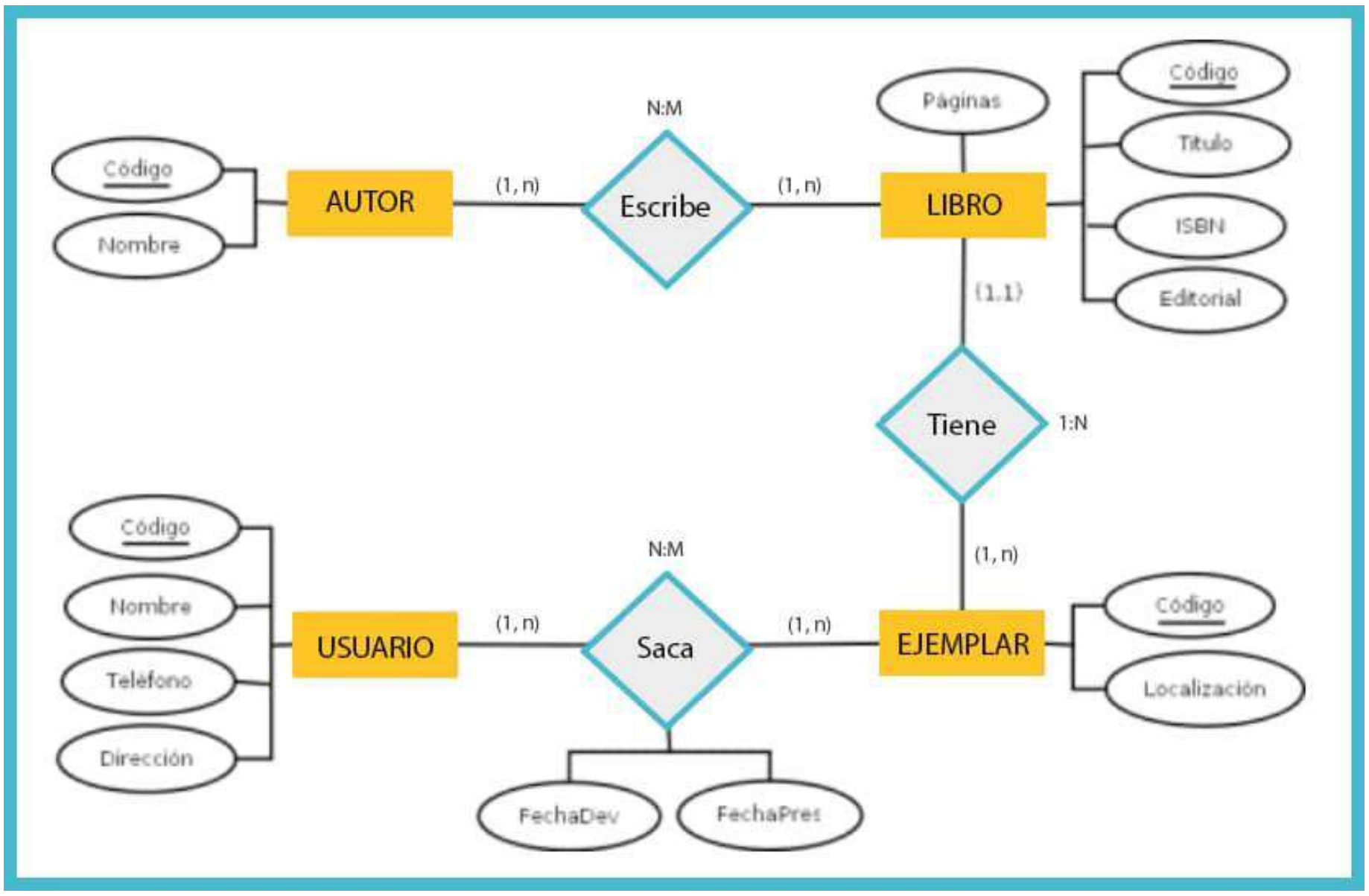
- **1 a 1**: Una persona *(e)* poseé *(r)* una única clave de estudiante *(e)* y viceversa.
- **1 a M**: Una factura *(e)* se emite *(r)* a una persona *(e)* y sólo a una, pero una persona *(e)* puede tener *(r)* varias facturas *(e)* emitidas a su nombre.
- **M a M**: Un cliente *(e)* puede comprar *(r)* varios productos *(e)* y un producto *(e)* puede ser comprado *(r)* por varios clientes *(e)*.

🔼 Regresar

Modelo Entidad-Relación

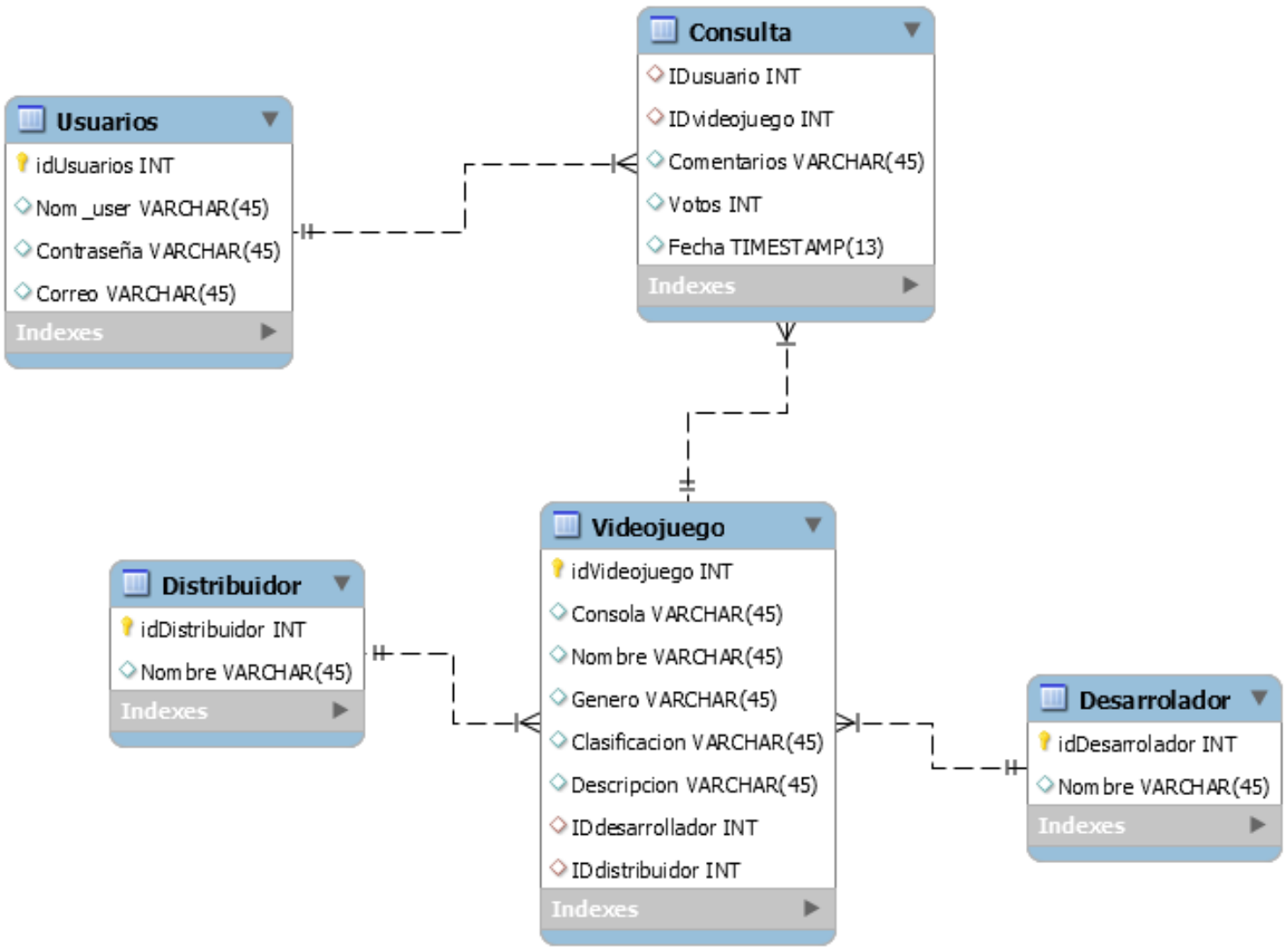
Un diagrama modelo entidad-relación es una herramienta para el modelo de datos, la cual facilita la representación de entidades de una base de datos.

Se caracteriza por utilizar una serie de símbolos y reglas para representar los datos y sus relaciones. Con este modelo conseguimos representar de manera gráfica la estructura lógica de una base de datos.



- Las entidades se representan con rectángulos.
- Los atributos se representan con óvalos que se conectan a la entidad a la que pertenecen.
- Los atributos que son llaves primarias se subrayan.
- Las relaciones se representan con rombos que conectan a las entidades relacionadas, dentro del rombo se coloca el verbo que hace la relación entre las entidades.

Hay una variante a este diagrama, que se llama **Modelo Relacional de la Base de Datos** que también ejemplifica gráficamente la relación de las entidades y la descripción de los atributos de estas.



Personalmente prefiero este tipo de diagrama por sobre el modelo entidad-relación, ya que nos permite describir el tipo de dato de cada atributo y se vuelve más fácil de manejar al tener cada entidad en una tabla con sus respectivos atributos.

Este tipo de diagramas lo puedes hacer con cualquier aplicación o *software* de diseño o diagramación, yo uso *Diagrams* que es gratuita.

🔼 Regresar

Normalización

La normalización de bases de datos es un proceso que se utiliza para organizar y optimizar la estructura de una base de datos para asegurar su integridad, evitar la redundancia y mejorar el rendimiento. La normalización consiste en la división de las entidades en varias entidades más pequeñas y relacionarlas mediante llaves foráneas.

La normalización se realiza a través de varios niveles o formas, cada uno de los cuales representa un grado de descomposición de la entidad original. Los tres niveles más comunes de normalización son la Primera Forma Normal (*1FN*), la Segunda Forma Normal (*2FN*) y la Tercera Forma Normal (*3FN*), aunque existen otros 2 niveles.

El objetivo de la normalización es reducir la redundancia y garantizar la integridad de los datos al asegurar que cada dato solo se almacene en un solo lugar y que los datos sean consistentes y coherentes. La normalización también ayuda a mejorar el rendimiento de la base de datos, ya que reduce el tamaño y la complejidad de las entidades, lo que facilita la indexación y la búsqueda de información.

Es importante tener en cuenta que la normalización puede tener un impacto en el rendimiento de la aplicación, ya que puede requerir una mayor cantidad de consultas y una complejidad adicional para

recuperar y manipular datos. Por lo tanto, es importante encontrar un equilibrio entre la normalización y la eficiencia en el diseño de la base de datos.

Formas Normales

Las formas normales son estándares para la organización y modelamiento de datos en una base de datos relacional. En total existen 5 formas normales.

1. **Primera Forma Normal (1FN)**: Cada atributo de una entidad debe contener solo valores atómicos, es decir, valores indivisibles que no pueden ser divididos en atributos más pequeños.
2. **Segunda Forma Normal (2FN)**: Además de cumplir con la *1FN*, cada atributo no dependiente funcionalmente de la llave principal debe estar en una entidad separada.
3. **Tercera Forma Normal (3FN)**: Además de cumplir con la *2FN*, todas las dependencias funcionales deben ser eliminadas, es decir, no deben existir dependencias funcionales transitorias.
4. **Cuarta Forma Normal (4FN)**: También llamada de Forma Normal de *Boyce-Codd* (FNBC), es una forma más restrictiva que la *3FN*, donde se garantiza que no existan dependencias funcionales parciales o transitivas en la entidad.
5. **Quinta Forma Normal (5FN)**: También conocida como Forma Normal de Domino-Clave (FNDC), en ella se debe garantizar que no haya dependencias múltiples de conjuntos en las entidades.

Al aplicar las formas normales a un modelo de base de datos, se puede asegurar que los datos sean consistentes, que no haya redundancia y que sea fácil de mantener y escalar.

Sin embargo, también es importante tener en cuenta que la aplicación de formas normales más rigurosas puede resultar en una estructura de base de datos más compleja y menos eficiente en términos de rendimiento. Por lo tanto, es importante encontrar un equilibrio entre la integridad de los datos y la eficiencia en el diseño de un modelo de base de datos.

- **Primera Forma Normal**: En la *1FN*, cada columna de una tabla debe contener únicamente valores atómicos, es decir, valores simples que no pueden ser divididos en partes más pequeñas.
- **Segunda Forma Normal**: La *2FN* requiere que cada columna no dependiente funcionalmente de la clave primaria de una tabla sea movida a una tabla separada. Esto significa que cada tabla debe representar un solo hecho o concepto.
- **Tercera Forma Normal**: La *3FN* requiere que todas las dependencias funcionales sean removidas de la tabla, es decir, que no haya redundancia de información.
- **Forma Normal de *Boyce-Codd***: La *FNBC* es una forma normal más rigurosa que la anteriores y requiere que cada dependencia funcional sea una clave candidata única.
- **Forma Normal de Dominio-Clave**: Esta forma normal (*FNDC*) es una extensiones de la *FNBC* y se utiliza para asegurar la integridad de los datos en modelos de datos más complejos. No debe haber dependencias funcionales múltiples, es decir, una dependencia funcional en la que varios atributos dependen de una clave externa.

Normalizando una base de datos

Veamos un ejemplo de normalización de base de datos.

Tenemos una entidad desnormalizada de **"Ventas"** de una tienda con la siguiente información:

Puedes normalizarme el siguiente modelo de datos

Venta	Fecha	Cliente	Correo	Teléfono	Dirección	Ciudad	País	Producto	Precio	Cantidad
1	01/01	Juan Perez	juan.perez@gmail.com	5512345678	Calle 1 No. 58-1 CP 03100	CDMX	México	Laptop	25,000.00	2
2	02/01	Pedro Gomez	pedro.gomez@gmail.com	3387654321	Calle 2 No. 85-6 CP 44100	Guadalajara	México	Celular	12,000.00	3
3	03/01	Ana Silva	ana.silva@gmail.com	8109128734	Calle 3 No. 33-3 CP 64000	Monterrey	México	Micrófono	2,500.00	1
4	04/01	Ana Silva	ana.silva@gmail.com	8109128734	Calle 3 No. 33-3 CP 64000	Monterrey	México	Laptop	25,000.00	1
5	05/01	Juan Perez	juan.perez@gmail.com	5512345678	Calle 4 45-3 03920	CDMX	México	Micrófono	2,500.00	3

La primera forma normal busca tener valores atómicos, es decir datos simples que no puedan ser divididos en parte más pequeñas, por lo que en el modelo anterior podríamos atomizar el nombre del cliente y su dirección quedando de la siguiente forma:

Venta	Fecha	Nombre	Apellido	Correo	Teléfono	Calle	Número	CP	Ciudad	País	Producto	Precio	Cantidad
1	01/01	Juan	Perez	juan.perez@gmail.com	5512345678	Calle 1	58-1	03100	CDMX	México	Laptop	25,000.00	2
2	02/01	Pedro	Gomez	pedro.gomez@gmail.com	3387654321	Calle 2	85-6	44100	Guadalajara	México	Celular	12,000.00	3
3	03/01	Ana	Silva	ana.silva@gmail.com	8109128734	Calle 3	33-3	64000	Monterrey	México	Micrófono	2,500.00	1
4	04/01	Ana	Silva	ana.silva@gmail.com	8109128734	Calle 3	33-3	64000	Monterrey	México	Laptop	25,000.00	1
5	05/01	Juan	Perez	juan.perez@gmail.com	5512345678	Calle 4	45-3	03920	CDMX	México	Micrófono	2,500.00	3

La segunda forma normal se refiere a la eliminación de las dependencias funcionales parciales. En este caso, podemos identificar que los datos del cliente se duplican en las ventas.

Por lo tanto, podemos crear una entidad separada llamada **"Clientes"** que almacene estos datos y en la entidad principal **"Ventas"** agregamos la llave foránea que haga referencia al cliente.

Venta	Fecha	Cliente	Producto	Precio	Cantidad
1	01/01	1	Laptop	25,000.00	2
2	02/01	2	Celular	12,000.00	3
3	03/01	3	Micrófono	2,500.00	1
4	04/01	3	Laptop	25,000.00	1
5	05/01	1	Micrófono	2,500.00	3

Cliente	Nombre	Apellido	Correo	Teléfono	Calle	Número	CP	Ciudad	País
1	Juan	Perez	juan.perez@gmail.com	5512345678	Calle 1	58-1	03100	CDMX	México
2	Pedro	Gomez	pedro.gomez@gmail.com	3387654321	Calle 2	85-6	44100	Guadalajara	México
3	Ana	Silva	ana.silva@gmail.com	8109128734	Calle 3	33-3	64000	Monterrey	México
1	Juan	Perez	juan.perez@gmail.com	5512345678	Calle 4	45-3	03920	CDMX	México

Sin embargo al extraer los datos del cliente se genera duplicidad de información, ya que se detecta que un cliente puede tener más de una dirección, por lo que es necesario crear una entidad separada llamada **"Direcciones"** que almacene estos datos y en la entidad principal **"Ventas"** agregamos la llave foránea que haga referencia a dicha dirección y finalmente la entidad **"Clientes"** sólo quedaría con la información personal de la persona.

Por lo que el modelo quedaría de la siguiente forma:

Venta	Fecha	Cliente	Dirección	Producto	Precio	Cantidad
1	01/01	1	1	Laptop	25,000.00	2
2	02/01	2	2	Celular	12,000.00	3
3	03/01	3	3	Micrófono	2,500.00	1
4	04/01	3	3	Laptop	25,000.00	1
5	05/01	1	4	Micrófono	2,500.00	3

Cliente	Nombre	Apellido	Correo	Teléfono
1	Juan	Perez	juan.perez@gmail.com	5512345678
2	Pedro	Gomez	pedro.gomez@gmail.com	3387654321
3	Ana	Silva	ana.silva@gmail.com	8109128734

Dirección	Cliente	Calle	Número	CP	Ciudad	País
1	1	Calle 1	58-1	03100	CDMX	México

Dirección	Cliente	Calle	Número	CP	Ciudad	País
2	2	Calle 2	85-6	44100	Guadalajara	México
3	3	Calle 3	33-3	64000	Monterrey	México
4	1	Calle 4	45-3	03920	CDMX	México

La tercer forma normal exige que no haya transparencias funcionales. Esto se logra removiendo todas las dependencias transitivas, es decir, aquellas dependencias en las que un atributo depende indirectamente de otro a través de un tercer atributo.

En este caso, la entidad **"Ventas"** ya está en la segunda forma normal, así que podemos continuar con la eliminación de dependencias transitivas.

La entidad **"Ventas"** depende transitoriamente del **"Producto"** a través de **"Precio"**. Por lo tanto, debemos crear una entidad adicional para los **"Productos"** que incluya la información de estos.

Por lo cual nuestro modelo quedaría de la siguiente forma:

Venta	Fecha	Cliente	Dirección	Producto	Cantidad
1	01/01	1	1	1	2
2	02/01	2	2	2	3
3	03/01	3	3	3	1
4	04/01	3	3	1	1
5	05/01	1	4	3	3

Producto	Nombre	Precio
1	Laptop	25,000.00
2	Celular	12,000.00
3	Micrófono	2,500.00

Cliente	Nombre	Apellido	Correo	Teléfono
1	Juan	Perez	juan.perez@gmail.com	5512345678
2	Pedro	Gomez	pedro.gomez@gmail.com	3387654321
3	Ana	Silva	ana.silva@gmail.com	8109128734

Dirección	Cliente	Calle	Número	CP	Ciudad	País
1	1	Calle 1	58-1	03100	CDMX	México
2	2	Calle 2	85-6	44100	Guadalajara	México
3	3	Calle 3	33-3	64000	Monterrey	México
4	1	Calle 4	45-3	03920	CDMX	México

La cuarta forma normal (*Boyce-Codd*), es más restrictiva con las dependencias transitivas, por lo que analizando la información del modelo detectamos que la entidad **"Direcciones"** sigue dependiendo del **"País"**, por lo que debemos crear una entidad adicional que contenga la información de dicho atributo.

Finalmente la quinta forma normal (Dominio-Clave) exige eliminar cualquier dependencia funcional múltiple, pero en este modelo no existen por lo que también cumple con esta última forma normal.

Al final de la normalización el modelo quedo de la siguiente manera:

Venta	Fecha	Cliente	Dirección	Producto	Cantidad
1	01/01	1	1	1	2
2	02/01	2	2	2	3
3	03/01	3	3	3	1
4	04/01	3	3	1	1
5	05/01	1	4	3	3

Producto	Nombre	Precio
1	Laptop	25,000.00
2	Celular	12,000.00
3	Micrófono	2,500.00

Cliente	Nombre	Apellido	Correo	Teléfono
1	Juan	Perez	juan.perez@gmail.com	5512345678
2	Pedro	Gomez	pedro.gomez@gmail.com	3387654321
3	Ana	Silva	ana.silva@gmail.com	8109128734

Dirección	Cliente	Calle	Número	CP	Ciudad	País
1	1	Calle 1	58-1	03100	CDMX	1
2	2	Calle 2	85-6	44100	Guadalajara	1
3	3	Calle 3	33-3	64000	Monterrey	1
4	1	Calle 4	45-3	03920	CDMX	1

País	Nombre	Dominio
1	México	mx

🔼 Regresar

Modelado de Datos

1. Identificar las entidades del sistema.
2. Identificar los atributos de las entidades.
3. Identificar las llaves primarias y foráneas.
4. Asignar una nomenclatura adeacuada a las entidades y sus atributos.
5. Identificar las entidades pivote del sistema.
6. Identificar los catálogos del sistema.
7. Identificar los tipos de relaciones del sistema.
8. Crear el Modelo Entidad-Relación del sistema.
9. Crear el Modelo Relacional de la base de datos del sistema.
10. Identificar los tipos de dato de los atributos de las entidades del sistema.
11. Identificar los atributos que puedan ser únicos en el sistema.
12. Identificar las reglas de negocio (Operaciones *CRUD*) del sistema.

🔼 Regresar

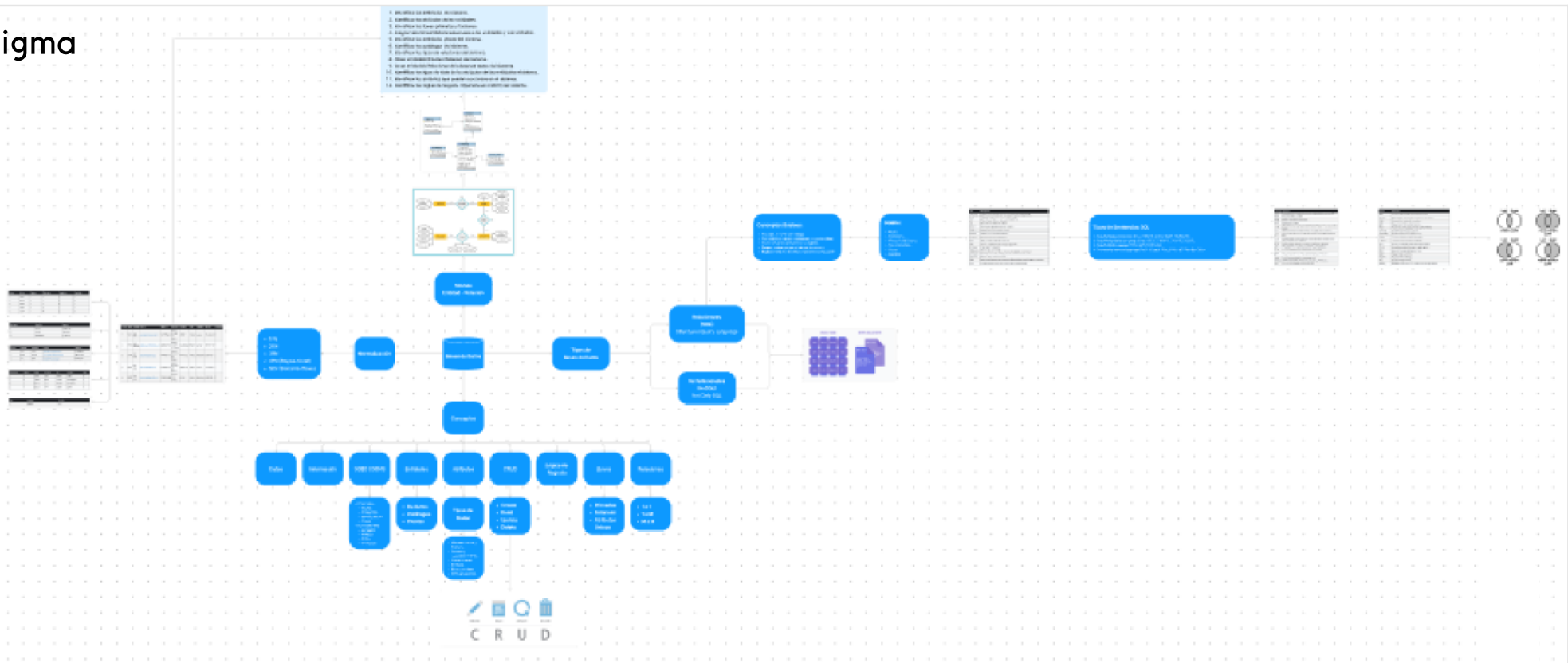
Aprende más

Si estás interesado en aprender más sobre bases de datos, no te pierdas mis cursos totalmente **gratuitos** en mi canal de *YouTube*.

¡¡¡Accede ya!!!

Ver Cursos

Figma



Bases de Datos Edited 2 months ago

🔼 Regresar