

Módulo 4 – Base de Datos

Índice

Introducción

Introducción a Bases de Datos	2
Organización de una base de datos	5

Bases de Datos

Base de Datos Relacional	9
Diseño de Base de Datos	22
Normalización de Datos.....	27

Programando Bases de Datos

Programando Bases de Datos	35
Conjuntos y JOINS en Bases de Datos	41
Transacciones en Base de Datos	51
Bloqueo o Locking en Bases de Datos	53

Introducción a Bases de Datos

¿Por qué Bases de Datos?

Las bases de datos surgen de la necesidad de las empresas en almacenar grandes cantidades de información de una forma rápida, sencilla y fiable, y que a su vez pudieran acceder a ella en cualquier momento sin necesidad de desplazarse a salas dedicadas a archivar documentación, como se hacía antiguamente cuando la tecnología de la información empezaba a emerger.

Cuando comenzó el despegue de los programas informáticos se empezaron a almacenar datos en los archivos de los programas, lo cual era más cómodo, pero aun así tenían grandes dificultades a la hora de querer modificar registros, estructuras o simplemente buscar información. Eran métodos lentos, propensos a errores de archivo e incómodos de implementar en grandes estructuras. Recordemos que estamos hablando de épocas donde no existía internet.

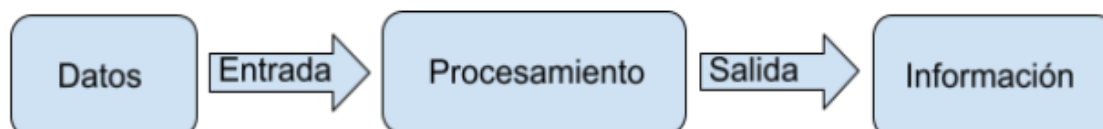
A finales de los años sesenta nacen las bases de datos. En estos sistemas se guardan los datos utilizados por los usuarios. Y los programas que consumían esos datos no se tenían que preocupar de su mantenimiento ni almacenaje, por lo que un cambio en la base de datos no tiene por qué afectar en principio a los programas que la utilizan.

Diferencia entre datos e información.

Todo sistema informático tiene como objetivo principal administrar y controlar los datos de manera eficiente para la toma de decisiones, por ello, dependiendo de lo que consideremos como dato deberíamos tenerlo en cuenta para nuestro sistema. Esto nos lleva a definir a los datos como todo aquello que se puede registrar en nuestra base de datos, que siendo procesado nos devolverá información relevante.

Podemos decir que, si nosotros tenemos un dato de temperatura, (por ejemplo 32°C), no nos dice nada por sí sólo, pero si obtenemos ese dato luego de consultar la temperatura de una ciudad determinada en determinado momento se convierte en información para conocer el clima de esa ciudad, a ese procesamiento de datos le llamamos información.

Un dato puede ser una letra, número, símbolo o palabra que por sí solo no tiene ningún significado, mientras que la información se define como un conjunto de datos procesados que tienen significado para el usuario.



¿Qué es una base de datos?

Como definición de base de datos entendemos que se trata de un conjunto de datos interrelacionados, almacenados sin redundancias innecesarias y que tienen un significado implícito, los cuales sirven a las aplicaciones sin estar relacionados de una manera directa entre ellos. Por otro lado, cuando hablamos de los datos, queremos decir: hechos conocidos que pueden registrarse y que tienen un significado implícito.

Cada día nos encontramos con actividades que requieren algún tipo de interacción con una base de datos (ingreso en un banco, reserva de una entrada para el teatro, solicitud de una suscripción a una revista, compra de productos, ...). Estas interacciones son ejemplos de lo que se llama aplicaciones tradicionales de bases de datos que básicamente almacenan información numérica o de texto, aunque los avances tecnológicos han permitido que también existan: bases de datos multimedia, sistemas de información geográfica (GIS), sistemas de proceso analítico on-line. Y ni hablar si tocamos el tema Big Data.

Una base de datos puede ser utilizada por varias aplicaciones y usuarios, y toda base de datos debe permitir insertar, modificar y borrar datos.

Hay dos grandes categorías o grupos de datos:

1. **Los datos de usuarios:** Datos usados por las aplicaciones
2. **Los datos de sistema:** Datos que la base de datos utiliza para su propia gestión como los usuarios registrados para operar la base, los privilegios de estos usuarios, configuraciones, etc

Un simple ejemplo de Base de Datos:

Una agenda con los nombres y teléfonos de un conjunto de personas conocidas es una base de datos, puesto que es una colección de datos relacionados con un significado implícito.

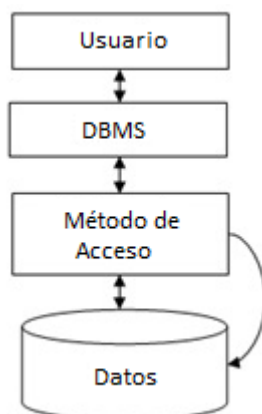
Este ejemplo de agenda de contactos hace referencia a dos elementos para que un conjunto de datos constituya una Base de Datos:

1. Relaciones entre datos, tema que se tratará en los capítulos siguientes.
2. Significado implícito de los datos que se atribuye dependiendo del contexto en que se utilizan los mismos. Por ejemplo, el dato fecha en una base de datos de VENTAS puede referirse a la fecha de emisión de las facturas, mientras que si la base de datos es de MÚSICA quizás corresponda a la fecha en que se grabó un tema musical. Es decir, el significado de un dato depende de la Base de Datos que lo contenga.

Para manipular y gestionar las bases de datos existen herramientas (software) denominadas sistemas gestores de bases de datos o [SGBD](#). Es decir, nos permite realizar las funciones de modificar, extraer y almacenar información de una base de datos, además de poseer herramientas con funciones de eliminar, modificar,

analizar, etc... datos de estas. Realiza la función concreta de interfaz entre la base de datos y los usuarios finales o los programas correspondientes, organizando los datos y permitiendo su acceso. En internet existen versiones gratuitas, de software libre y software propietario.

En el siguiente esquema podemos ver cómo el usuario interactúa con los datos a través del DBMS o SGBD:



Esquema DBMS. (n.d.).

Otro simple ejemplo de Base de Datos:

Cuando trabajamos con programas de hojas de cálculo como Excel, LibreOffice, OpenOffice y otros. En la práctica muchas veces los usamos como base de datos. La estructura de filas y columnas en donde vamos listando datos, de forma ordenada, estructurada y relacionada termina siendo una tarea de indexación de datos. Además, si sumamos la posibilidad de editar online como Google Spreadsheets, agregamos la posibilidad de que varios usuarios o programas, usen de forma simultánea la información para leer o editar.

Seguramente, en algún momento, has usado el Excel o Google Spreadsheets como base de datos de facturas, horas, o algún otro tipo de datos.

En la siguiente imagen observamos una simple tabla para llevar un control de horas trabajadas de un desarrollador freelance. Bien podría ser considerada su base de datos de horas trabajadas:

	A	B	C	
1	Número de Ticket	Descripción	Horas trabajadas	
2		132 Login no debe ser case sensitive		2
3		144 Cambiar color de botones		1
4		156 Bug con cálculo de kilometros		5
5		177 Crear función de Notificaciones Push		12
6		178 Refactorizar componente de perfil		5
7		170 Refactorizar componente de registro		6
8		190 Agrega pantalla de about		1
9				
10				

Organización de una base de datos

Toda base de datos organizada debe cumplir los siguientes objetivos:

- **Tiene que ser versátil:** esto quiere decir que, dependiendo de los usuarios o las aplicaciones, puedan hacer diferentes cosas o traten a los datos de formas distintas.
- **Tiene que atender con la rapidez adecuada**
- **Tiene que tener un índice de redundancia lo más bajo posible.**
- **Tiene que disponer de una alta capacidad de acceso** para ganar el mayor tiempo posible en la realización de consultas.
- **Tener un alto índice de integridad**, esto significa que al tener muchos usuarios consultando y escribiendo en una misma base de datos no puede haber fallos en la inserción de datos, errores por redundancia o lenta actualización.

Tipos de Base de Datos

Hay varios tipos y enfoques de cómo organizar y transaccionar los datos. Nos enfocaremos en las bases de datos Relacionales, pero a los fines informativos dejamos una breve descripción de las otras opciones que existen:

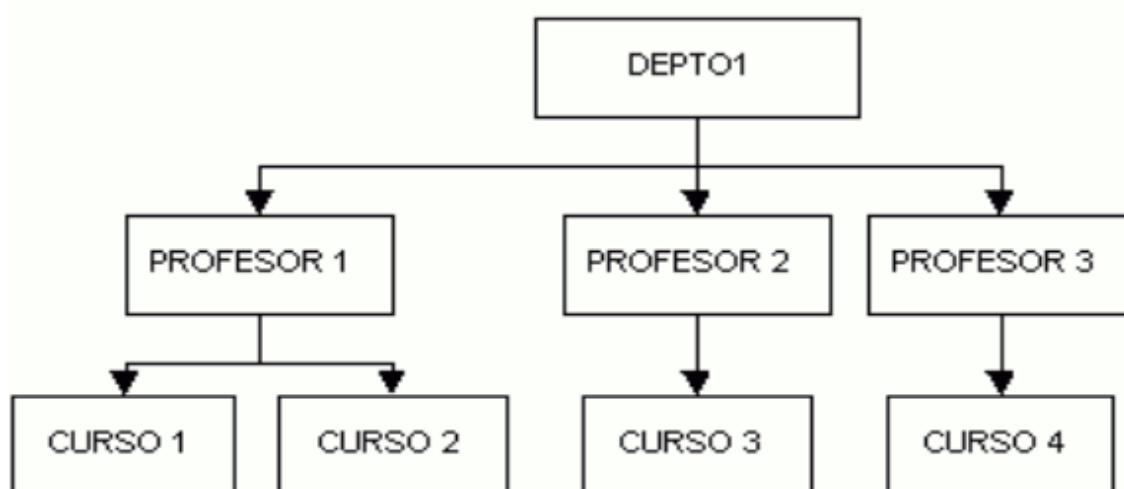
Según su variabilidad

Se encuentran las bases de datos estáticas que son aquellas que sus datos no pueden modificarse, están diseñadas especialmente para la lectura de sus datos. Por lo general se suelen utilizar para almacenar datos históricos, que no cambiarán, para realizar proyecciones estadísticas y ayudar a la toma de decisiones. Por otro lado, las bases de datos dinámicas son aquellas que sus datos se pueden actualizar, ya sea agregando, modificando o eliminando los mismos durante el transcurso del tiempo.

Según el contenido

Según el contenido que almacenan las bases de datos se pueden clasificar de la siguiente forma:

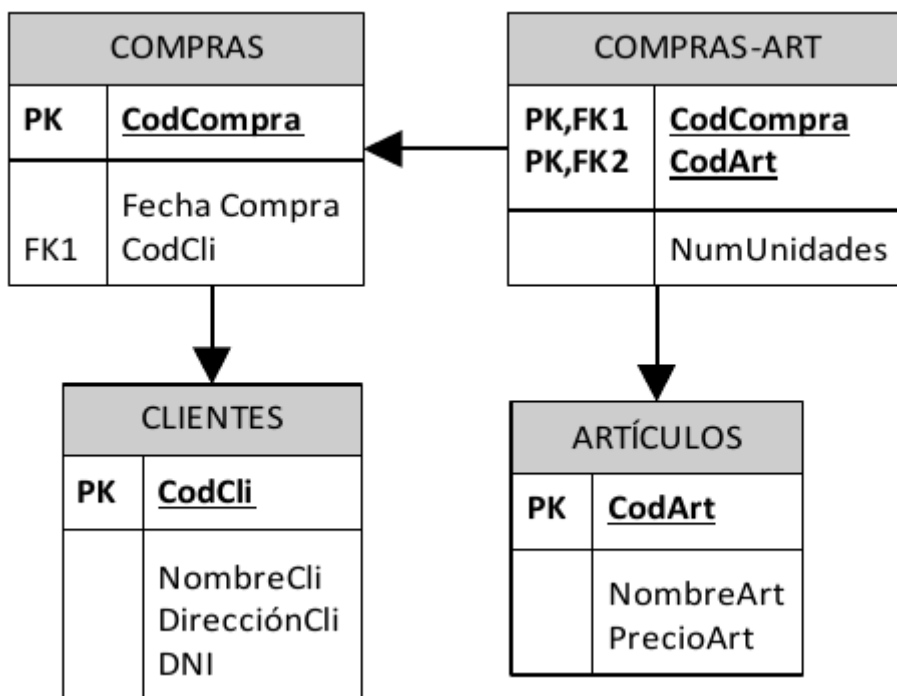
Bases de datos jerárquicas: La información se organiza en nodos jerárquicos. Desde un nodo raíz, cada nodo puede contener información de varios hijos y así sucesivamente. Son utilizadas en sistemas con gran volumen de datos y datos compartidos. Un ejemplo de cómo se organizan los datos se muestra en la siguiente imagen:



Base de Datos Jerárquica. (n.d.).

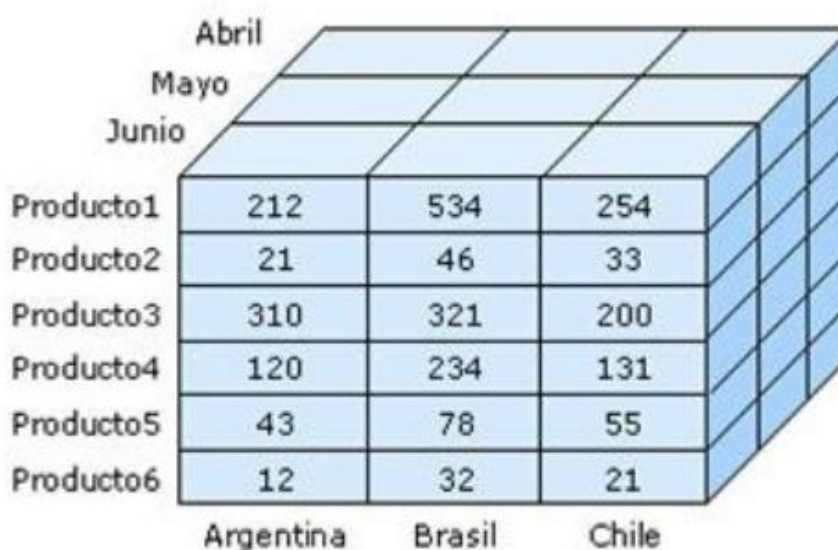
Transaccionales: El diseño de la base de datos está encaminado a recoger las diferentes transacciones que se producen en un sistema. El diseño e implementación están orientados a la rapidez y seguridad de las transacciones, y a la consistencia y durabilidad de los datos. Se deben crear sistemas totalmente fiables, que permitan la recuperación de los errores (revirtiendo los procesos que sean necesarios) y que aíslen las diferentes fuentes que acceden al programa para evitar errores e incoherencias.

Relacionales: Se basan en la relación de datos estructurados, fiables y homogéneos. Los datos están relacionados conceptualmente, no por su utilización y su implementación en máquina. Las diferentes entidades del sistema son accesibles en tiempo real y compartidas por los usuarios. El centro de los modelos relacionales son las entidades y las relaciones entre ellas, pudiendo haber relaciones normales, de herencia, composición, etc. Todas ellas basadas en las relaciones que se producen en el mundo real de las entidades lógicas.



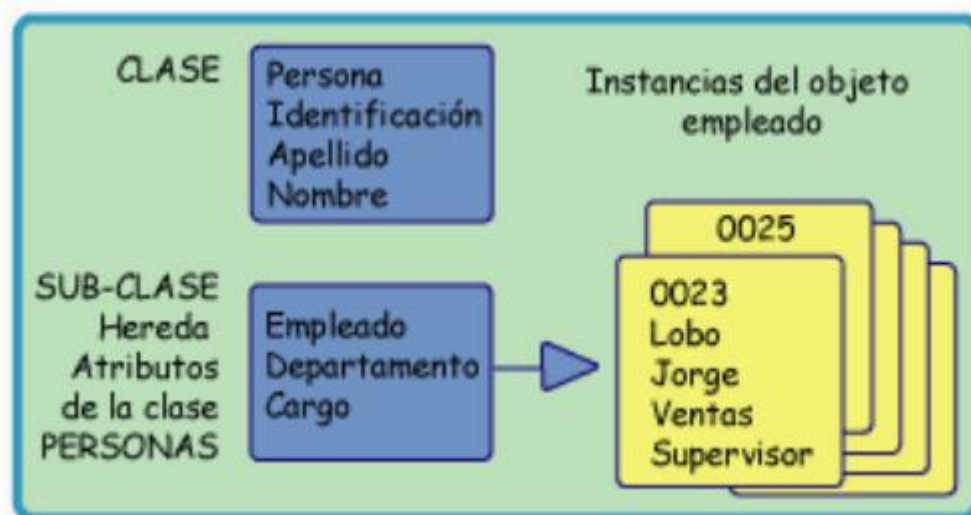
Base de datos relacional. (n.d.).

Multidimensionales (BDMD): La información de la base de datos puede verse contenida en una sola tabla multivaluada donde se almacenan registros referidos a las dimensiones o métricas que se van a analizar. Estas tablas se asimilan a un hipercubo, las dimensiones de los cubos se corresponden con la tabla y el valor almacenado en cada celda equivale al de la métrica.



Base de datos multidimensional. (n.d.).

Orientadas a objetos: Están basadas en el concepto de objeto de programación. Tradicionalmente, las bases de datos almacenaban los datos y sus relaciones y los procedimientos se almacenaban en los programas de aplicación. En estas bases de datos, sin embargo, se combinan los procedimientos de una entidad con sus datos.



Base de Datos Documental. (n.d.). [Ilustrator].

Clave-valor: Basadas en el concepto de las [tablas hash](#). Se basan en almacenar una serie de registros clave-valor. La clave es un string convencional mientras que los valores pueden ser desde un string hasta una lista o un conjunto. Está diseñada para almacenar grandes cantidades de información, que puede almacenarse en sesión o caché, para ser compartida por varios servidores. Un ejemplo de este tipo de bases de datos es [Redis](#).

Orientadas a columnas: La principal diferencia es que los registros no se organizan en filas sino en columnas. Todo el dominio de valores de un caso de uso, por ejemplo, «Nombre de persona», se pueden acceder como si fueran una única unidad. Cambia totalmente el paradigma de la consulta. Tradicionalmente se recorrían todas las filas de una o varias tablas para devolver los campos seleccionados que cumplieran cierta selección, ahora se seleccionan unos pocos registros de los que nos traemos todo el dominio de valores que cumpla las condiciones. Esto tiene sentido en consultas analíticas.

Orientada a documentos o NoSQL: Cada registro corresponde a un documento. Estos documentos se diferencian de los registros de las bases de datos SQL en que se autodefinen ellos mismos, es decir, cada documento define el formato que va a tener, son libres de esquemas (diferente número de campos, campos de longitud variable, campos multivalor, etc.). Los documentos comparten entre sí una parte de información similar y otra muy diferente y se identifican por una clave única, que puede ser desde un string hasta una URI. Las consultas suelen ser dinámicas y muy dispares, normalmente, se dispone de un API o lenguaje de

interrogación para las consultas de documentos según el contenido de los mismos. Este tipo de consultas son muy útiles de cara a los análisis estadísticos. Ejemplos de bases de datos documentales son MongoDB, CouchDB, ArangoDB, etc.

Base de datos SQL vs NoSQL

En la actualidad se habla mucho de las bases de datos NoSQL como MongoDB. Es normal que nos preguntemos sobre sus comparativas con las SQL. Vamos a realizar una tabla comparativa para resumir esta comparación:

Característica	Base de datos SQL	Base de datos NoSQL
Desempeño (performance)	Bajo	Alto
Availability (disponibilidad)	Pobre	Buena
Fiabilidad (reliability)	Buena	Pobre
Consistencia (consistency)	Buena	Pobre
Almacenamiento (data storage)	Tamaño medio	Optimizado para grandes datos
Escalabilidad (scalability)	Alta pero cara	Alta

Base de Datos Relacional

Tras ser postulados sus fundamentos en 1970 por [Edgar Frank Codd](#), de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos. Su idea fundamental es el uso de "relaciones". Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "tuplas". Pese a que ésta es la teoría de las bases de datos relacionales creadas por Edgar Frank Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar. Esto es pensando en cada relación como si fuese una tabla que está compuesta por registros (las filas de una tabla), que representan las tuplas, y campos (las columnas de una tabla).

En este modelo, el lugar y la forma en que se almacenan los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario esporádico de la base de datos. Los datos pueden ser recuperados o almacenados mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar los datos.

El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, Structured Query Language o Lenguaje Estructurado de Consultas, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

Ventajas de una base de datos relacional

Sus ventajas:

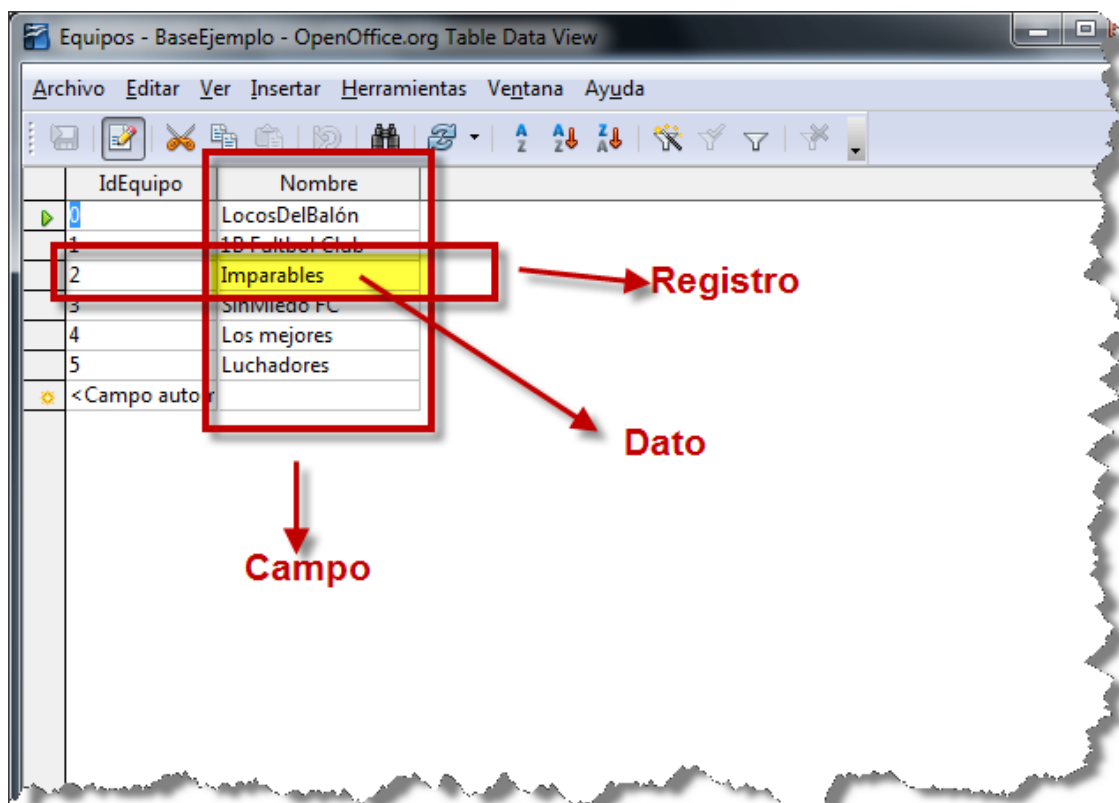
1. Menor redundancia. No hace falta tanta repetición de datos. Aunque, sólo los buenos diseños de datos tienen poca redundancia.
2. Menor espacio de almacenamiento. Gracias a una mejor estructuración de los datos.
3. Acceso a los datos más eficiente. La organización de los datos produce un resultado más óptimo en rendimiento.
4. Datos más documentados. Gracias a los metadatos que permiten describir la información de la base de datos.
5. Independencia de los datos y los programas y procesos. Esto permite modificar los datos sin modificar el código de las aplicaciones.
6. Integridad de los datos. Mayor dificultad de perder los datos o de realizar incoherencias con ellos.
7. Mayor seguridad en los datos. Al limitar el acceso a ciertos usuarios.

Como contrapartida encontramos los siguientes inconvenientes:

1. Instalación costosa. El control y administración de bases de datos requiere de un software y hardware potente.
2. Requiere personal cualificado. Debido a la dificultad de manejo de este tipo de sistemas.
3. Implantación larga y difícil. Debido a los puntos anteriores. La adaptación del personal es mucho más complicada y lleva bastante tiempo.

Tablas, Filas y Columnas

Las Base de Datos relacional nos permiten agrupar los datos en forma de tablas que, a su vez, dentro de las tablas, los datos se organizan en filas y columnas. La intersección de un registro (fila) y un campo (columna) nos da el dato. Como fue el primer ejemplo que dimos, esto nos recuerda mucho a la forma en que trabajamos con las planillas de cálculo como Excel, LibreOffice, OpenOffice, etc. En una planilla de cálculo lo podemos visualizar de la siguiente forma:



Dato.

Cuando hagamos consultas a la base de datos, seleccionaremos tablas, y le pediremos al motor que busque los registros realizando alguna lógica sobre las columnas o campos como por ejemplo, si tomamos la primera imagen de ejemplo:

	A	B	C	
1	Número de Ticket	Descripción	Horas trabajadas	
2	132	Login no debe ser case sensitive	2	
3	144	Cambiar color de botones	1	
4	156	Bug con cálculo de kilómetros	5	
5	177	Crear función de Notificaciones Push	12	
6	178	Refactorizar componente de perfil	5	
7	170	Refactorizar componente de registro	6	
8	190	Agrega pantalla de about	1	
9				
10				

Podríamos solicitarle a la base de datos que nos de todos los registros de “Números de Ticket” que tenga más de 10 horas trabajadas. Así será que la consulta nos deberá responder que es la fila 5 con 12 horas trabajadas.

Modelos de datos

Un modelo de datos es una serie de conceptos que se utilizan para describir un conjunto de datos y las operaciones para manipularlos.

Hay dos tipos de modelos de datos: los modelos conceptuales y los modelos lógicos. Los modelos conceptuales se utilizan para representar la realidad a un alto nivel de abstracción. Mediante los modelos conceptuales se puede construir una descripción de la realidad fácil de entender. En los modelos lógicos, las descripciones de los datos tienen una correspondencia sencilla con la estructura física de la base de datos.

En el diseño de bases de datos se usan primero los modelos conceptuales para lograr una descripción de alto nivel de la realidad, y luego se transforma el esquema conceptual en un esquema lógico. El motivo de realizar estas dos etapas es la dificultad de abstraer la estructura de una base de datos que presente cierta complejidad. Un esquema es un conjunto de representaciones lingüísticas o gráficas que describen la estructura de los datos de interés. Los modelos conceptuales deben ser buenas herramientas para representar la realidad, por lo que deben poseer las siguientes cualidades:

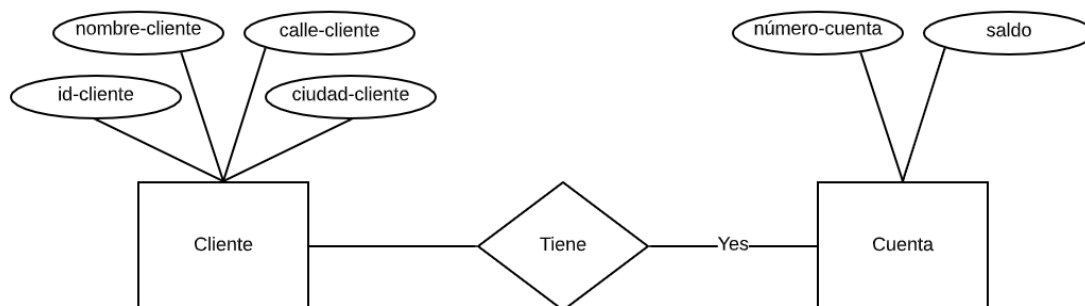
- **Expresividad:** deben tener suficientes conceptos para expresar perfectamente la realidad.
- **Simplicidad:** deben ser simples para que los esquemas sean fáciles de entender.
- **Minimalidad:** cada concepto debe tener un significado distinto.
- **Formalidad:** todos los conceptos deben tener una interpretación única, precisa y bien definida. En general, un modelo no es capaz de expresar todas las propiedades de una realidad determinada, por lo que hay que añadir aserciones que complementen el esquema.

Modelo Entidad-Relación

Cuando empezamos a pensar o diseñar una base de datos, puede parecer simple en un principio, pero en la medida que nuestra idea de base de datos crece, el diseño también crece y necesitamos una forma de poder conceptualizar y visualizar todos los elementos que la componen, cómo se relacionan y qué características tienen.

Entonces denominado por sus siglas como: E-R, este modelo representación de la realidad a través de entidades, que son objetos que existen y que se distinguen de otros por sus características, por ejemplo: un alumno se distingue de otro por sus características particulares como lo es el nombre, o el número de control asignado al entrar a una institución educativa, así mismo, un empleado, una materia, etc.

La siguiente imagen ilustra un DER:



Modelo Entidad Relación

Hay tres elementos básicos en un modelo ER:

1. Entidad
2. Atributo
3. Relación

Entidad: Una entidad representa una "cosa", "objeto" o "concepto" del mundo real.

Algunos ejemplos:

- Una persona: se diferencia de cualquier otra persona, incluso siendo gemelos.
- Un automóvil: aunque sean de la misma marca, el mismo modelo, etc, tendrán atributos diferentes, por ejemplo, el número de chasis.
- Una casa: aunque sea exactamente igual a otra, aún se diferenciará en su dirección

Atributos: Los atributos son las características que definen o identifican a una entidad. como, por ejemplo, para una entidad Persona tenemos: Edad, género, peso, altura, etc.

Veamos otro ejemplo: Consideremos una empresa que requiere controlar a los vendedores y las ventas que ellos realizan; de este problema determinamos que los objetos o entidades principales a estudiar son el empleado (vendedor) y el artículo (que es el producto en venta), y las características que los identifican son:

- Empleado: Nombre, Puesto, Salario, Numero de Empleado
- Artículo: Nombre Descripción Costo

Conjunto de relaciones: Consiste en una colección, o conjunto, de relaciones entre las entidades. Por ejemplo: Dadas las entidades "Habitación" y "Huésped", la relación que podemos tener es que uno o más huéspedes estarán alojados en una

habitación. Además, a esta relación le podemos dar un título: “Se aloja” ya que los huéspedes se alojan en las habitaciones.

Las relaciones entre las entidades se pueden agrupar en 3 tipos según su finalidad.

Cardinalidad de las relaciones

La cardinalidad de un atributo indica el número mínimo y el número máximo de valores que puede tomar para cada ocurrencia de la entidad o relación a la que pertenece. El tipo de cardinalidad se representa mediante una etiqueta en el exterior de la relación, respectivamente: "1:1", "1:N" y "N:M".

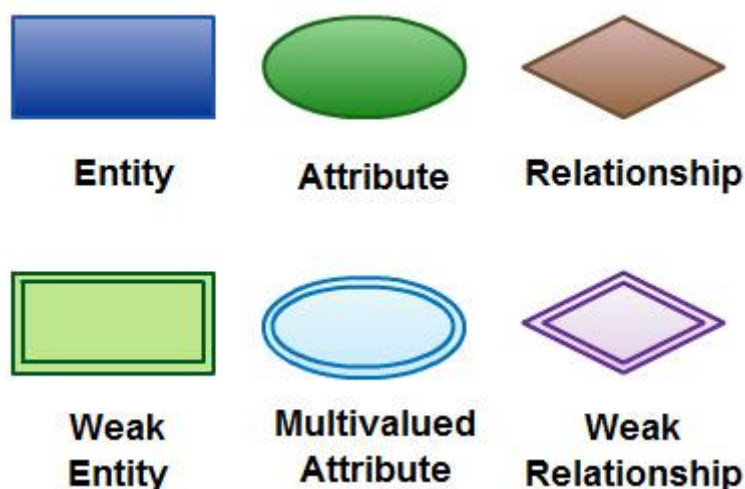
- **Uno a Uno (1:1)** se da cuando un elemento de una entidad se puede relacionar solamente con un solo registro de otra entidad y viceversa.
- **Uno a Muchos (1:M)** se da cuando un registro de una entidad A se puede relacionar con cero o muchos registros de otra entidad B y cada registro de la entidad B se relaciona con un sólo registro de la entidad A.
- **Muchos a Muchos (N:M)** se da cuando un registro de una entidad se relaciona con cero o varios registros de otra entidad

Si tuviéramos el caso de una base de datos de una escuela podríamos pensar en las siguientes relaciones:

1. Un alumno de una escuela puede estar solamente en un solo curso (1:1)
2. En una materia de la escuela, un alumno tendrá varias notas durante el año (1:N)
3. En la universidad, un alumno puede estar en muchas clases, y las clases pueden tener alumnos de diferentes carreras (N:N)

Símbolos y notaciones de diagramas ER

Las entidades, sus atributos y las relaciones se pueden representar de forma gráfica a través de algunos elementos visuales. A continuación, una imagen que nos describe en primer término cómo se visualiza una entidad, un atributo y una relación:



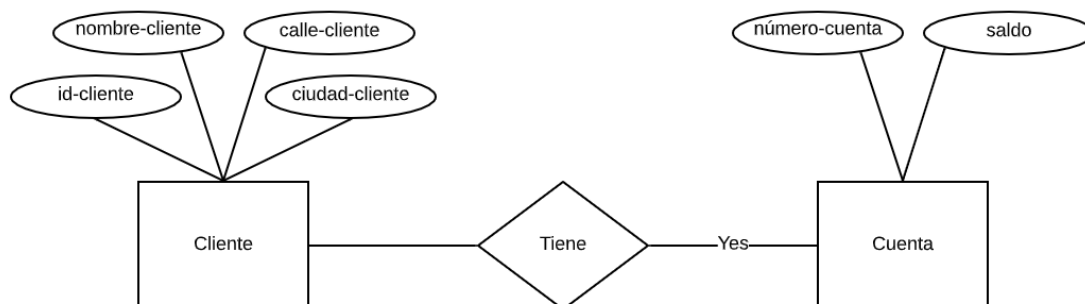
Símbolos y notaciones de diagramas ER

Hay más elementos que se basan en los elementos principales, que son las entidades débiles, los atributos multivalorados, atributos derivados, relaciones débiles y relaciones recursivas. A los fines prácticos solamente nos vamos a centrar en la simbología básica, pero se debe saber que existen más símbolos que brindar un diagrama más específico. Al final de la sección dejamos 4 tablas adicionales con descripciones más detalladas para las entidades, las relaciones, los atributos y las cardinalidades.

Caso de aplicación de un DER.




Se nos pide que armemos una base de datos para llevar las cuentas de los clientes. Los datos que necesitamos almacenar de los clientes son el nombre, la ciudad y la calle donde vive, y un número de identificación del cliente. Por otro lado, las cuentas de los clientes tienen un número de cuenta y un saldo.



Si bien en la próxima sección veremos cómo diseñar el DER, al ser ésta una herramienta muy intuitiva, podemos tomar la siguiente imagen y entender las partes ya vistas.




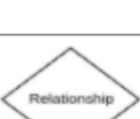








Elementos adicionales de un DER

Cómo mencionamos dejamos unas tablas adicionales de los diagramas de entidades, relaciones, atributos y cardinalidades a los fines de conocer que existen y se pueden utilizar o encontrar en otros DER más complejos.

Símbolo de entidad	Nombre	Descripción
	Entidad fuerte	Estas figuras son independientes de otras entidades y con frecuencia se les denomina entidades matriz ya que a menudo tienen entidades débiles que dependen de ellas. También tendrán una clave primaria, que distinga a cada suceso de la entidad.
	Entidad débil	Las entidades débiles dependen de algún otro tipo de entidad. No tienen claves primarias y no tienen significado en el diagrama sin su entidad matriz.
	Entidad asociativa	Las entidades asociativas relacionan las instancias de varios tipos de entidades. También contienen atributos que son específicos a la relación entre esas instancias de entidades.

Símbolo de relación	Nombre	Descripción
	Relación	Las relaciones son asociaciones entre dos o más entidades.
	Relación débil	Las relaciones débiles son conexiones entre una entidad débil y su propietario.

Símbolo de atributo	Nombre	Descripción
	Atributo	Los atributos son las características de una entidad, una relación de muchos a muchos, o una relación de uno a uno.
	Atributo de varios valores	Los atributos de valores múltiples son aquellos que pueden tomar más de un valor.
	Atributo derivado	Los atributos derivados son atributos cuyos valores se pueden calcular a partir de valores de atributos relacionados.
	Relación	Las relaciones son asociaciones entre dos o más entidades.

Símbolo de cardinalidad	nombre
	One
	Many
	One (and only one)
	Zero or one
	One or many
	Zero or many

Diagramas DER

Cómo hacer un Diagrama Entidad-Relación o DER

Los DER pueden verse confusos, pero no son tan complejos como parecen. Los diagramas entidad-relación se pueden generar fácilmente, y existen muchas herramientas que permiten facilitarnos la tarea como:

- <https://app.creately.com/>
- <https://www.lucidchart.com/>
- <https://app.diagrams.net/>

Para continuar, crearemos un diagrama ER conceptual de un sistema simple en el cual un estudiante se registra para un curso que es impartido por un profesor.

Identificar los componentes

1. Determina las entidades: Las entidades generalmente son sustantivos como auto, banco, estudiante o producto.

En este ejemplo, las tres entidades son “Estudiante”, “Curso” y “Profesor”.



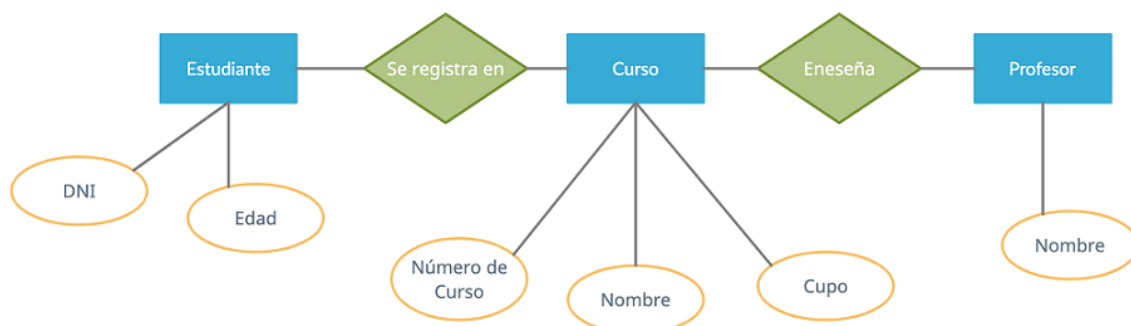
2. Identifica las relaciones: Las relaciones resaltan cómo las entidades interactúan entre sí.

Las relaciones generalmente son verbos como “compra”, “contiene” o “hace”. En nuestro ejemplo, las relaciones “Se registra en” y “Enseña” explican de forma efectiva las interacciones entre las tres entidades.



3. Agrega atributos: Los atributos muestran características específicas de una entidad, detallando qué información es importante para el modelo.

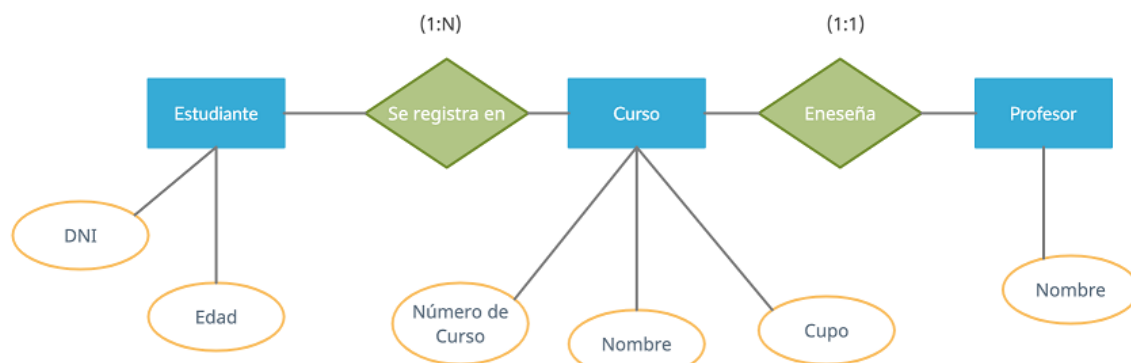
En un diagrama ER, los atributos son necesarios para modelar qué características se incluirán con cada entidad. Siguiendo el ejemplo, los estudiantes tienen un DNI y una edad, los cursos tienen un número de curso, una cantidad de asientos disponibles y el nombre del curso. Los profesores por su parte solamente nos interesa guardar el nombre.



4. Cardinalidad de las relaciones

Para finalizar nos quedaría preguntarnos cómo es la cardinalidad entre los estudiantes.

- En un curso puede haber muchos estudiantes, pero un estudiante solo puede estar en un curso. (1:N)
- En este caso particular, la institución nos exige que solamente pueda haber un profesor por curso, y que cada profesor solamente puede tener asignado un curso.

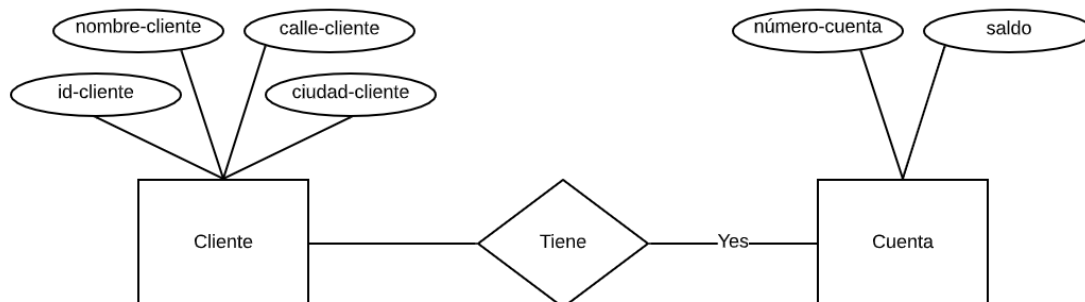


5. Completa el diagrama

Es increíblemente importante organizar el diagrama ER de una forma lógica para aumentar la comprensión. El propósito principal de los diagramas de entidad-relación es modelar una base de datos compleja, por lo que es esencial aprender cómo crear diagramas ER simples y lógicos.

Del DER a las especificaciones

Ahora hagamos el camino inverso. Si tuviéramos el siguiente DER ¿Qué podríamos decir?



Modelo Entidad Relación

Primero nos damos cuenta de que es una base de datos de algún tipo de banco o empresa, y luego podríamos empezar a desmenuzar sus entidades, atributos y relaciones.

- Las entidades son Clientes y Cuentas.
- Los atributos de Clientes son: Nombre, calle, ciudad y un id
- Los atributos de la cuenta son número de cuenta y saldo.
- La relación es que los clientes tienen cuentas

Ahora en este DER faltan las cardinalidades. ¿Vos qué dirías al respecto?

Hagamos un siguiente paso en la comprensión del DER y volquemos estas dos entidades del DER, cliente y cuenta a una tabla para poder visualizarlo de la siguiente forma:

Cliente			
id-cliente	nombre-cliente	calle-cliente	ciudad-cliente
1	Juan	San Martín 200	Bahía Blanca
2	Pedro	Alsina 442	Miramar
Cuentas			
numero-cuenta	id-cliente	saldo	
335456	1	-200	
235477	1	100	

Tenemos una entidad “Cliente” donde vamos almacenando los datos del cliente, y una entidad “Cuentas” donde, además de almacenar los datos de la cuenta, necesariamente debe existir un dato que nos permita conectar ambas tablas.

En principio todos los DER podríamos esquematizarlos en tablas. Sin embargo, a medida que aumentan las relaciones se vuelve más complejo, pero veremos otras

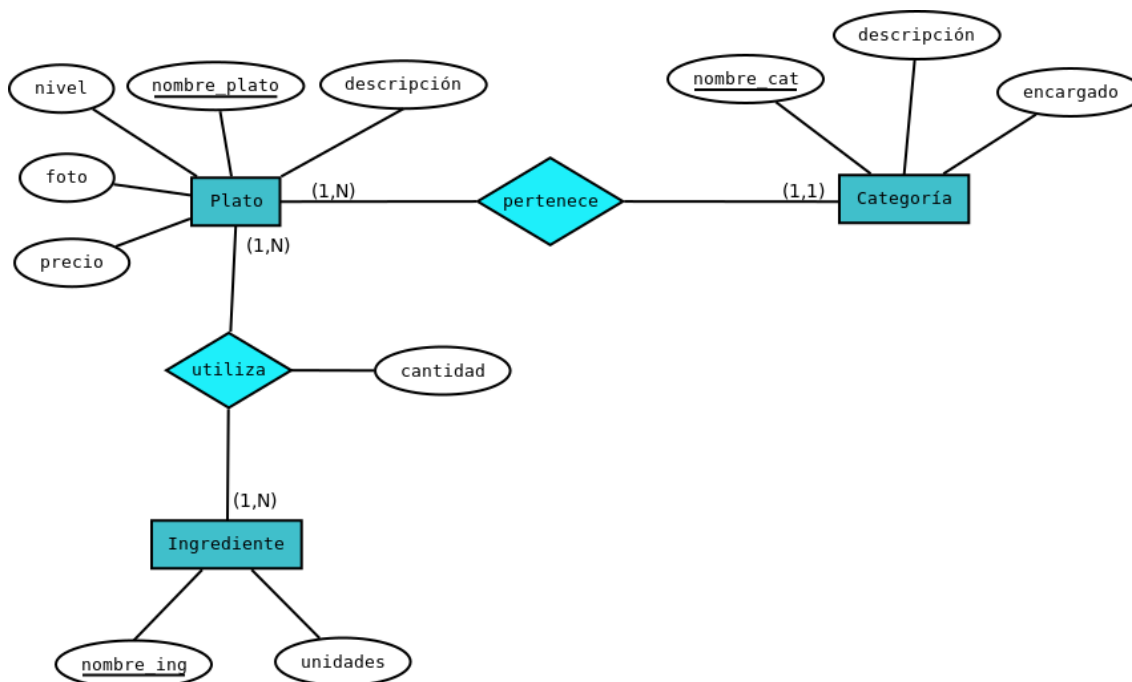
formas de esquematizarlos. Lo importante es no perder de vista que las entidades, en una base de datos relacional, las podemos ver como tablas de filas y columnas.

Ejemplo carta de un restaurante:

El siguiente diseño consiste en construir una base de datos que almacene la carta de un restaurante.

- Para cada plato, se desea obtener su nombre, descripción, nivel de dificultad (de elaboración), una foto y el precio final para el cliente.
- Cada plato pertenece a una categoría.
- Las categorías se caracterizan por su nombre, una breve descripción y el nombre del encargado.
- De los platos, se desea conocer las recetas para su realización, con la lista de ingredientes necesarios, aportando la cantidad requerida, las unidades de medida (gramos, litros, etc.) y cantidad actual en el almacén.

En la siguiente imagen se representa el DER de la base de datos anteriormente solicitada:



Ejemplo DER

Beneficios de los DER

Los Diagramas de Entidad-Relación constituyen un marco muy útil para crear y manipular bases de datos.

En primer lugar, los DER son fáciles de comprender y no requieren que una persona reciba una capacitación exhaustiva para poder trabajar con ellos de manera eficiente y precisa. Esto significa que los diseñadores pueden utilizar los DER para comunicarse fácilmente con los desarrolladores, los clientes y los usuarios finales, independientemente de su competencia en TI.

En segundo lugar, los DER son fácilmente traducibles en tablas relacionales que pueden utilizarse para construir rápidamente bases de datos. Además, los DER pueden ser utilizados directamente por los desarrolladores de bases de datos como el plano para implementar los datos en aplicaciones de software específicas.

Por último, los DER pueden aplicarse en otros contextos, como la descripción de las diferentes relaciones y operaciones dentro de una organización.

Desafío Portfolio Web

En este momento estamos en sobradas condiciones de desarrollar el DER para nuestro propio Portfolio Web. Empecemos a pensar cuales son las entidades que necesitaremos tener, sus relaciones, atributos y cardinalidades. Es importante que lo empecemos a hacer y luego le iremos agregando más capas de información.

Diseño de Base de Datos

Cuando diseñamos nuestros sistemas, pensamos en todo lo que quisiéramos que estuviera almacenado en una base de datos y diseñamos la base de datos para guardar dichos datos. Pero debemos de ser realistas acerca de nuestras necesidades y decidir qué dato es realmente necesario. Frecuentemente podemos generar algunos datos sobre la marcha sin tener que almacenarlos en una tabla de una base de datos. En estos casos también tiene sentido hacer esto desde el punto de vista del desarrollo de la aplicación. Un modelo de datos es básicamente una "descripción" de algo conocido como contenedor de datos (algo en donde se guardan los datos), así como de los métodos para almacenar y recuperar datos de esos contenedores. Los modelos de datos no son cosas físicas: son abstracciones que permiten la implementación de un sistema eficiente de base de datos.

Metodología de diseño de bases de datos

El diseño de una base de datos es un proceso complejo que abarca decisiones a muy distintos niveles. La complejidad se controla mejor si se descompone el problema en subproblemas y se resuelve cada uno de estos subproblemas independientemente, utilizando técnicas específicas. Así, el diseño de una base de datos se descompone en diseño conceptual, diseño lógico y diseño físico.

El esquema anterior, es una representación lógica que puede ser fácilmente convertido a código de base de datos.

Por ejemplo, la tabla Clientes quedaría con el siguiente código SQL:

```
CREATE TABLE Clientes(  
    Id_Cliente int NOT NULL,  
    Nombre varchar(50) NOT NULL,  
    Apellido varchar(50) NOT NULL,  
    Id_Localidad int NOT NULL);  
)
```

Diseño físico

El diseño físico parte del esquema lógico y da como resultado un esquema físico. Un esquema físico es una descripción de la implementación de una base de datos en memoria secundaria: las estructuras de almacenamiento y los métodos utilizados para tener un acceso eficiente a los datos. Por ello, el diseño físico depende del DBMS concreto y el esquema físico se expresa mediante su lenguaje de definición de datos.

Algunos de los elementos del diseño físico son los índices.

Tipos de Datos

Conocer y definir los tipos de datos en el diseño físico es muy importante porque forman parte de las restricciones que sirven para establecer las reglas de una tabla. Especificando qué tipos de datos se pueden añadir en un registro evita que se inserten datos incorrectos. Es necesario conocer los tipos de datos de SQL. A continuación, una sencilla tabla de tipos de datos que manipulan las bases de datos.

Tipo de Datos	Longitud	Descripción
BIT	1 byte	Valores Si/No - True/False - Verdadero/Falso
DATETIME	8 bytes	Fecha u hora entre los años 100 y 9999.
DOUBLE	8 bytes	Un valor en punto flotante de precisión doble con un rango de - 1.79769313486232*10308 a -4.94065645841247*10-324 para valores negativos siendo para valores positivos entre 4.94065645841247*10-324 a 1.79769313486232*10308 y 0.
LONG	4 bytes	Valor entero largo entre -2,147,483,648 y 2,147,483,647.
LONGTEXT	1 byte por carácter	De cero a un máximo de 1.2 gigabytes.
TEXT	1 byte por carácter	De cero a 255 caracteres.

Índice

Para facilitar la obtención de información de una tabla se utilizan índices. El índice de una tabla desempeña la misma función que el índice de un libro: permite encontrar datos rápidamente. En el caso de las tablas, localiza registros.

Una tabla se indexa por uno o varios campos, y posibilita el acceso directo y rápido haciendo más eficiente las búsquedas. Sin índice, se debe recorrer secuencialmente toda la tabla para encontrar un registro consumiendo más tiempo y recursos. Entonces el objetivo de un índice es acelerar la recuperación de información. La desventaja es que consume espacio en el disco ya que requiere realizar una tarea de indexación. Es una técnica que optimiza el acceso a los datos, mejora el rendimiento acelerando las consultas y otras operaciones. Es útil cuando la tabla contiene miles de registros. Es importante identificar el o los campos por los que sería útil crear un índice, aquellos campos por los cuales se realizan operaciones de búsqueda con frecuencia.

Hay distintos tipos de índices, a saber:

- **Index:** Puede haber varios por tabla. Es un índice común. Los valores no necesariamente son únicos y aceptan valores nulos. Podemos darle un nombre, si no se lo damos, se coloca uno por defecto. "key" es sinónimo de "index".
- **Unique:** Es un índice para los cuales los valores deben ser únicos y diferentes, aparece un mensaje de error si intentamos agregar un registro

con un valor ya existente. Permite valores nulos y pueden definirse varios por tabla.

- **Clave Primaria (PK)**

Clave Primaria (PK)

Una clave primaria es la columna o colección de columnas que nos permiten identificar de forma única a una fila determinada en una tabla. La clave primaria proporciona una forma importante de distinguir una fila de otra. Subrayar las columnas o la colección de columnas que componen la clave primaria usualmente es la mejor forma de representar la clave primaria de cada tabla de la base de datos.

Veamos un ejemplo, si tenemos una tabla con datos de personas, el número de documento puede establecerse como clave primaria, es un valor que no se repite; puede haber personas con igual apellido y nombre, incluso el mismo domicilio (padre e hijo, por ejemplo), pero su documento será siempre distinto.

Existen tres tipos de claves primarias;

1. **Clave natural** es una clave primaria compuesta de una columna que identifica de forma única a una entidad, por ejemplo el DNI de una persona o la patente de un auto.
2. **Clave artificial** es una columna creada para una entidad con el propósito de servir únicamente como clave primaria y es visible para los usuarios.
3. **Clave subrogada** es una clave primaria generada por el sistema, usualmente un tipo de datos generado automáticamente que suele estar escondido del usuario. Normalmente se define con el nombre de "id" y se define como un valor autoincremental. Esto hace que, al insertar un nuevo registro, este valor aumente.

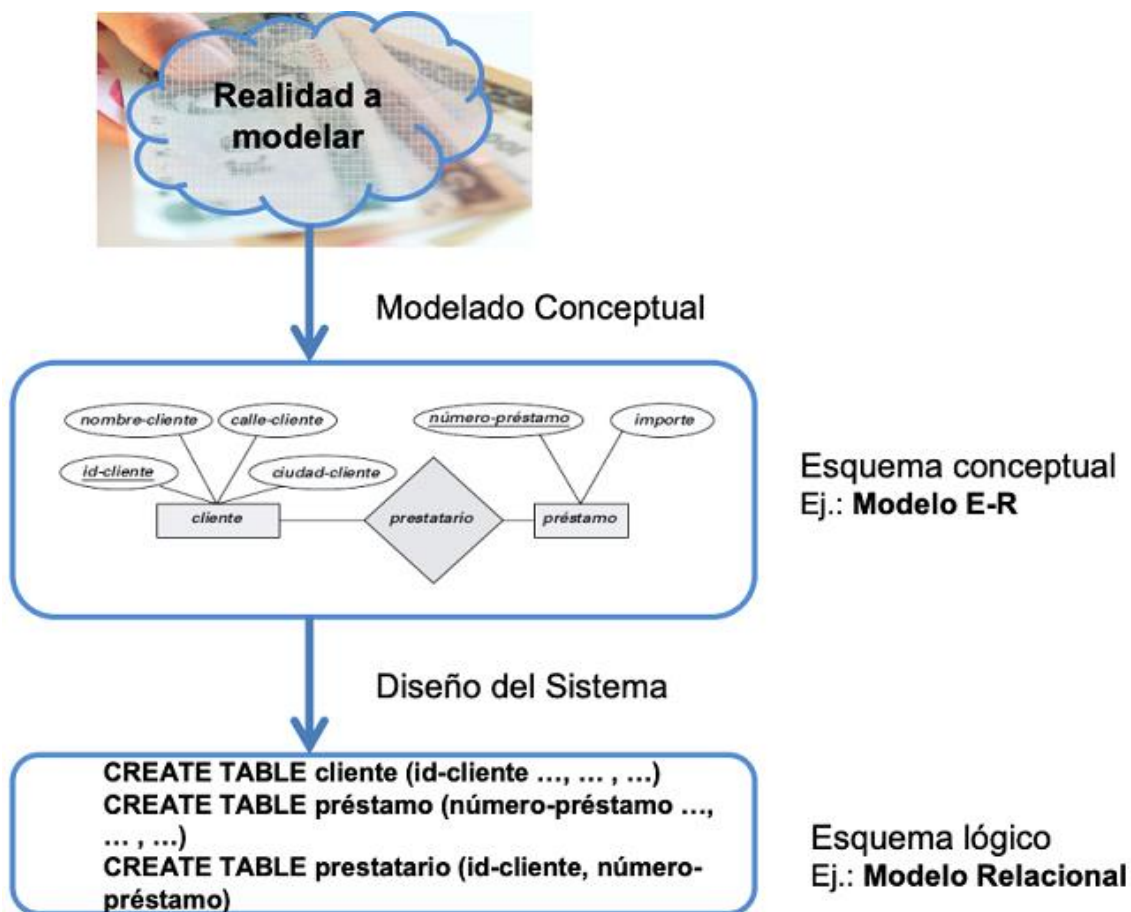
Clave Foránea (FK)

Una clave foránea es un campo o colección de campos de una tabla cuyos valores deben coincidir con los valores de la clave primaria de una segunda tabla.

De la realidad a la base de Datos

En resumen, el ejercicio de diseñar una base de datos parte de un proceso de 4 pasos:

1. Observación y análisis de la realidad de un problema.
2. Tomar el análisis y hacer un modelado conceptual mediante un DER.
3. Con el DER como mapa de ruta, se realiza el diseño lógico.
4. En este punto la base de datos ya se encuentra construida y nos queda el diseño físico que nos permitirá optimizar las consultas y aplicar restricciones para garantizar la integridad de la información y de la relación de datos.



Ejemplo Modelo DER. (n.d.)

Normalización de Datos

La normalización es la técnica de estandarización y validación de datos que se utiliza para diseñar las tablas y establecer las relaciones entre ellas. Esto nos permite una mayor organización para eliminar la redundancia de datos y proteger la integridad de los mismos. Sin la normalización, tendríamos datos redundantes los cuales traen aparejados problemas de mantenimiento, además de ocupar espacio en el disco duro, como también, dependencias incoherentes.

Las formas normales se pueden clasificar en:

1. Primera Forma Normal (1FN)
2. Segunda Forma Normal (2FN)
3. Tercera Forma Normal (3FN)
4. Cuarta Forma Normal (4FN)
5. Quinta Forma Normal (5FN)

Vamos a realizar el proceso de normalización sobre la siguiente tabla:

Id_Cliente	Nombre	Apellido	Email	Provincia	Localidad	CVU	EstadoCuenta
1	JORGE	GOMEZ	jorge@test.com; jgomez@test.com	SALTA	EL JARDIN	122334455	HABILITADA
2	LUISA	PEREZ	luisaperez@test.com	CÓRDOBA	CARLOS PAZ	212223334	HABILITADA
3	MARTHA	GONZALES	martha@test.com	TUCUMAN	TAFI VIEJO	312233445	HABILITADA
4	JUAN	PEREZ	juan@test.com	CÓRDOBA	LA FALDA	185274196	HABILITADA
5	PASCUAL	VARGAS	vargasp@test.com; pvargas@hotmail.com	CÓRDOBA	LA FALDA	321654789	HABILITADA

Aclaración: CVU es Clave virtual uniforme.

En esta tabla, que es la normalidad en la vida cotidiana de las personas, para los desarrolladores es casi una pesadilla de datos.

Podemos encontrar los siguientes problemas:

1. Datos duplicados como nombres de provincia, estados de cuenta, etc
2. Campos con varios datos
3. Datos relacionados
4. Podríamos tener errores en la relación de Provincias y Localidades. Por algún error una localidad pudo haber quedado junto a una provincia que no le correspondía.
5. Por algún error humano, alguien pudo haber escrito mal el nombre de una provincia o localidad. Encontrar y solucionar en la base de datos ese incidente puede ser muy incómodo
6. No tenemos control si hay emails o CVUs duplicados
7. Los procesos de búsqueda y actualización son ineficientes. Por ejemplo, si quisiéramos buscar y reemplazar el email de un usuario como Pascual Vargas.

Realicemos el proceso de normalización y veamos qué va ocurriendo con cada uno de estos problemas.

Primera Forma Normal (1FN)

Una tabla está en primera forma normal cuando todos los datos son atómicos, es decir, cada atributo tiene su propia celda y todas las columnas contienen el mismo tipo de datos. Sirve para eliminar los grupos repetidos.

Como vemos en el siguiente ejemplo, la primera forma normal no se cumple por el campo Email. En la primera y en la quinta fila tenemos dos emails:

Id_Cliente	Nombre	Apellido	Email	Provincia	Localidad	CVU	EstadoCuenta
1	JORGE	GOMEZ	jorge@test.com; jgomez@test.com	SALTA	EL JARDIN	122334455	HABILITADA
2	LUISA	PEREZ	luisaperez@test.com	CÓRDOBA	CARLOS PAZ	212223334	HABILITADA
3	MARTHA	GONZALES	martha@test.com	TUCUMAN	TAFI VIEJO	312233445	HABILITADA
4	JUAN	PEREZ	juan@test.com	CÓRDOBA	LA FALDA	185274196	HABILITADA
5	PASCUAL	VARGAS	vargasp@test.com; pvargas@hotmail.com	CÓRDOBA	LA FALDA	321654789	HABILITADA

Para solucionarlo, podemos crear un registro por cada email o eliminar el atributo que no cumple dicha condición. En la siguiente imagen vemos cómo quedaría la tabla si separamos el email y creamos un nuevo registro para cumplir con la condición. Ahora tenemos un email por registro:

Id_Cliente	Nombre	Apellido	Email	Provincia	Localidad	CVU	EstadoCuenta
1	JORGE	GOMEZ	jgomez@test.com	SALTA	EL JARDIN	122334455	HABILITADA
1	JORGE	GOMEZ	jorge@test.com	SALTA	EL JARDIN	122334455	HABILITADA
2	LUISA	PEREZ	luisaperez@test.com	CÓRDOBA	CARLOS PAZ	212223334	HABILITADA
3	MARTHA	GONZALES	martha@test.com	TUCUMAN	TAFI VIEJO	312233445	HABILITADA
4	JUAN	PEREZ	juan@test.com	CÓRDOBA	LA FALDA	185274196	HABILITADA
5	PASCUAL	VARGAS	pvargas@hotmail.com	CÓRDOBA	LA FALDA	321654789	HABILITADA
5	PASCUAL	VARGAS	vargasp@test.com	CÓRDOBA	LA FALDA	321654789	HABILITADA

Sin embargo, se nos presenta un nuevo problema. Vemos que JORGE GOMEZ aparece dos veces, y lo mismo ocurre con PASCUAL VARGAS.

Entonces para cumplir con la primera forma normal eliminando el atributo email, tendremos que crear una tabla emails que contenga las columnas email, que funcionara como clave primaria (PK), y una columna Id_Cliente, que funcionara como Clave Foránea que apunta hacia el id del cliente.

Cientes

Id_Cliente	Nombre	Apellido	Provincia	Localidad	CVU	EstadoCuenta
1	JORGE	GOMEZ	SALTA	EL JARDIN	122334455	HABILITADA
2	LUISA	PEREZ	CÓRDOBA	CARLOS PAZ	212223334	HABILITADA
3	MARTHA	GONZALES	TUCUMAN	TAFI VIEJO	312233445	HABILITADA
4	JUAN	PEREZ	CÓRDOBA	LA FALDA	185274196	HABILITADA
5	PASCUAL	VARGAS	CÓRDOBA	LA FALDA	321654789	HABILITADA

Emails

Email	Id_Cliente
jorge@test.com	1
lgomez@test.com	1
luisaperez@test.com	2
martha@test.com	3
juan@test.com	4
vargasp@test.com	5
pvargas@hotmail.com	5

El resultado final es que tenemos una tabla con todos los datos de los usuarios, y una segunda tabla que nos permite relacionar los emails con los datos de usuario. Así si un usuario tiene N emails, en esta nueva tabla tendremos N mails relacionados con el mismo id de usuario.

En este primer paso hemos eliminado algunos de los problemas mencionados anteriormente:

1. Campos con varios datos. Cada columna ahora contiene un único valor
2. Los procesos de búsqueda y actualización son ineficientes. Por ejemplo, si quisiéramos buscar reemplazar el email de un usuario como Pascual Vargas.
 1. El sistema es eficiente para buscar usuario por email
 2. Si queremos editar, agregar o borrar el email de un usuario solamente tenemos que realizar la operación sobre la tabla Emails

Segunda Forma Normal (2FN)

Para estar en la segunda forma normal, se agregan atributos que no forman parte de ninguna clave a las condiciones de la primera forma normal. Su funcionamiento depende de la clave primaria. Ésta sirve para eliminar los datos redundantes.

Después de pasar por el proceso de 1FN la tabla de clientes nos quedó de la siguiente forma

Cientes

Id_Cliente	Nombre	Apellido	Provincia	Localidad	CVU	EstadoCuenta
1	JORGE	GOMEZ	SALTA	EL JARDIN	122334455	HABILITADA
2	LUISA	PEREZ	CÓRDOBA	CARLOS PAZ	212223334	HABILITADA
3	MARTHA	GONZALES	TUCUMAN	TAFI VIEJO	312233445	HABILITADA
4	JUAN	PEREZ	CÓRDOBA	LA FALDA	185274196	HABILITADA
5	PASCUAL	VARGAS	CÓRDOBA	LA FALDA	321654789	HABILITADA

El problema que vamos a resolver es la falta de normalización de las columnas Provincia y Localidad.

Para empezar, será necesario:

1. Crear las tablas Provincias y Localidades
2. Eliminar el la columna Provincia de la tabla Cientes

¿Por qué eliminamos la columna provincia? Porque el dato de la provincia lo podemos obtener sabiendo la localidad. Todas las localidades pertenecen a una única provincia, así es que, si se sabe la localidad, se sabe la provincia.

Entonces, ahora obtendremos la segunda forma normal, ya que obtendremos la provincia desde la tabla localidades y la localidad desde la tabla clientes. Además, creamos la tabla EstadosCuentas con Id_Estado (PK) y Nombre.

Al aplicar la segunda forma normal nos quedarían las tablas Cientes, Provincias y Localidades:

Provincias

Id_Provincia	Nombre
1	CÓRDOBA
2	SALTA
3	TUCUMAN

Localidades

Id_Localidad	Nombre	Id_Provincia
1	EL JARDÍN	2
2	CARLOS PAZ	1
3	TAFI VIEJO	3
4	LA FALDA	1

Cientes

Id_Cliente	Nombre	Apellido	Id_Localidad	CVU	Id_EstadoCuenta
1	JORGE	GOMEZ	1	122334455	1
2	LUISA	PEREZ	2	212223334	1
3	MARTHA	GONZALES	3	312233445	1
4	JUAN	PEREZ	4	185274196	1
5	PASCUAL	VARGAS	4	321654789	1

Observemos la lógica de estas 3 tablas:

1. Tenemos una tabla Provincias con el nombre de la provincia y su respectivos id.
2. Una tabla Localidades donde cada localidad tiene su id, el nombre de la localidad y un id de provincia que se corresponde con la tabla Provincias
3. Tabla Cientes donde ya no tenemos la columna provincias, y solamente tenemos la columna Localidad con el id de la localidad.

Ahora volvamos a los problemas que mencionamos anteriormente

1. Podríamos tener errores en la relación de Provincias y Localidades. Por algún error una localidad pudo haber quedado junto a una provincia que no le correspondía. Este problema ya no existe, todas las relaciones de provincia y localidades existen en una única tabla, y de ser necesario cambiar una relación la tarea se simplifica a editar el registro de una única tabla.
1. Por algún error humano, alguien pudo haber escrito mal el nombre de una provincia o localidad. Encontrar y solucionar en la base de datos ese incidente puede ser muy incómodo. Nuevamente este problema se soluciona editando un único registro, en una única tabla.

La importancia de una buena normalización es que además de prevenir errores y ser eficiente en el uso del sistema, nos ayuda a realizar tareas de mantenimiento mucho más simples.

Para mantener la consistencia del proceso de normalización, aprovechamos y separamos los estados de cuenta en una tabla aparte. Si no te diste cuenta, en la

tabla clientes ahora hay una columna Id_Estado_Cuentas y en la siguiente imagen esta es su correspondiente tabla de relaciones:

EstadosCuentas

Id_Estado	Nombre
1	HABILITADA
2	DESHABILITADA

Tercera Forma Normal (3FN)

Para cumplirse la tercera forma normal, a las dos condiciones anteriores hay que agregarle que los atributos que no son clave no pueden depender de manera transitiva de una clave candidata. Aquí se deben eliminar las columnas que no dependen de la clave principal.

Ésta forma normal es considerada como el estándar a cumplir por los esquemas relacionales.

Siguiendo el ejemplo, tenemos que agregar la tabla Cuentas la cual tiene los atributos CVU (PK), Id_Estado e Id_Cliente y en la tabla Clientes eliminaremos a columna CVU y Id_EstadoCuenta quedando las tablas de la siguiente forma:

EstadosCuentas

Id_Estado	Nombre
1	HABILITADA
2	DESHABILITADA

Provincias

Id_Provincia	Nombre
1	CÓRDOBA
2	SALTA
3	TUCUMAN

Localidades

Id_Localidad	Nombre	Id_Provincia
1	EL JARDÍN	2
2	CARLOS PAZ	1
3	TAFI VIEJO	3
4	LA FALDA	1

Cuentas

CVU	Id_Estado	Id_Cliente
122334455	1	1
212223334	1	2
312233445	1	3
185274196	1	4
321654789	1	5

Clientes

Id_Cliente	Nombre	Apellido	Id_Localidad
1	JORGE	GOMEZ	1
2	LUISA	PEREZ	2
3	MARTHA	GONZALES	3
4	JUAN	PEREZ	4
5	PASCUAL	VARGAS	4

Con el resultado obtenido, en la tabla Cuentas ahora tenemos la posibilidad de mantener la información del CVU y el estado de la cuenta de una forma más eficiente y desacoplada de la tabla Clientes. La cual se ha reducido significativamente a los datos que son relevantes para los detalles de un cliente como nombre, apellido y localidad.

En este punto podemos observar que una única tabla que teníamos al inicio se ha convertido en 5 tablas. En cada tabla encontramos datos representativos, de forma ordenada y relacionada.

Cuarta Forma Normal (4FN)

El objetivo de la cuarta forma normal es evitar que nuestras tablas posean dependencias multivaluadas. Las dependencias multivaluadas son aquellas donde la existencia de dos o más relaciones N:M (Muchos a muchos) causando redundancia.

Como nuestro ejemplo de momento no cuenta con ninguna instancia de relación de muchos a muchos (N:M) tenemos que asumir que necesitamos habilitar los movimientos que tiene cada cliente con uno o más tipos de moneda.

Para esto crearemos la tabla Monedas, la cual tendría los atributos Id_Moneda y Nombre, y una tabla ClientesMonedas que tendrá las columnas Id_Cliente e Id_Moneda, en donde asignaremos con qué tipos de monedas puede operar cada cliente:

Monedas

Id_Moneda	Nombre
1	PESO
2	DOLAR
3	BITCOIN

ClientesMonedas

Id_Cliente	Id_Moneda
1	1
1	2
2	1
2	3
3	1
3	2
4	1
5	1
5	2
5	3

Cómo vemos en la tabla anterior, ahora nuestros clientes tienen la posibilidad de manejar diferentes tipos de moneda. En la tabla ClientesMoneda vemos que un

cliente puede tener muchas monedas, y que una moneda puede ser usada por muchos clientes (N:M).

Quinta Forma Normal (5FN o forma normal de Proyección-Unión)

Una relación se encuentra en 5FN si ya pasó por la normalización 4FN y además, las dependencias existentes son las dependencias de Join o de unión con sus proyecciones. Esto ocurre cuando podemos obtener la tabla original por medio de la unión de sus proyecciones, relacionándose entre sí a través de la clave primaria o alguna de sus claves alternativas.

Esto quiere decir que si mediante la unión de los campos de las tablas, podemos recuperar la tabla inicial. Si hacemos el ejercicio de ir conectando todos los id de las tablas, veremos que es posible reconstruir la tabla inicial.

Programando Bases de Datos

Si queremos crear, insertar o consultar datos desde una base de datos será necesario interactuar con el motor de la base de datos mediante lenguaje de consulta estructurado o SQL. SQL es un lenguaje de computación que se asemeja al inglés, pero que comprenden los programas de base de datos.

Si comprendemos el funcionamiento de SQL podremos crear mejores consultas además de facilitar la forma de solucionar una consulta que no devuelve los resultados que queremos.

¿Qué es SQL?

SQL es un lenguaje de computación para trabajar con conjuntos de datos y las relaciones entre ellos. Los programas de bases de datos relacionales, como SQL Server, MySQL, MariaDB, etc. usan SQL para trabajar con datos. El lenguaje SQL no es difícil de leer y entender, incluso para un usuario inexperto resulta algo intuitivo. Al igual que muchos lenguajes de computación, SQL es un estándar internacional reconocido por organismos de estándares como [ISO](#) y [ANSI](#).

Comandos en SQL

Muchas de las acciones como crear tabla e insertar datos se pueden hacer de forma visual a través de un Sistema Gestor de Base de Datos, es importante conocer las principales acciones que podemos hacer con SQL para luego poder realizar consultas y desarrollos más complejos. Los siguientes comandos los podremos ir probando en el SQL Server, pero también podemos hacer pruebas más sencillas con intérpretes online de SQL como los siguientes:

- <http://sqlfiddle.com/>
- <https://www.jdoodle.com/execute-sql-online/>

Como ocurre con todo lenguaje, existe una documentación en la cual podemos encontrar mucha más información y detalle. Siempre es importante consultar más detalles.

Crear una Base de Datos

Todo comienza con la siguiente instrucción

```
CREATE DATABASE MI_BASE_DE_DATOS
```

Crear una tabla

Una vez que tenemos una base de datos podemos empezar a crear nuestras tablas o entidades. Es importante en este momento conocer los tipos de datos:

```
CREATE TABLE Nombre_Tabla(  
  Nombre_Campo tipo_valor opciones,  
  Nombre_Campo_2 tipo_valor opciones  
);
```

Si quisiéramos crear una tabla de Clientes sería por ejemplo así:

```
CREATE TABLE Clientes(  
  Id_Cliente int NOT NULL,  
  Nombre varchar(50) NOT NULL,  
  Apellido varchar(50) NOT NULL,  
  Id_Localidad int NOT NULL  
);
```

Crear Clave Primaria (PK)

Para crear una clave primaria, fundamental en todas las tablas ejecutamos el siguiente comando:


```
ALTER TABLE Nombre_Tabla  
ADD PRIMARY KEY (Nombre_campo);
```

Si lo aplicamos a nuestro ejemplo de la tabla de clientes nos quedaría de la siguiente forma:

```
ALTER TABLE Clientes  
ADD PRIMARY KEY (Id_Cliente);
```

Crear Clave Foránea (FK)

Para crear una clave foránea recordemos que necesitamos tener una segunda tabla y utilizamos un campo que nos permita relacionar ambas tablas:

```
ALTER TABLE Tabla_origen ADD FOREIGN KEY(campo_tabla_origen)  
REFERENCES Tabla_destino (campo_clave_primaria_destino)
```

Aplicado a nuestro ejemplo anterior de clientes, supongamos que tenemos una segunda tabla con los datos de las localidades. Para crear la clave foránea sería:

```
ALTER TABLE Clientes ADD FOREIGN KEY(Id_Localidad)  
REFERENCES Localidades (Id_Localidad)
```

Seleccionando datos

Una de las sentencias más usadas es el SELECT que sirve para obtener los datos de una tabla. Si queremos seleccionar todos los campos de una tabla aplicamos la siguiente sentencia:

```
SELECT * FROM Nombre_tabla
```

Si queremos seleccionar algunos campos, solamente debemos listarlos separados por coma:

```
SELECT campo_1,campo_2,... FROM Nombre_tabla
```

En nuestro caso si quisiéramos obtener la lista de Nombre y Apellidos de la tabla de Clientes nos quedaría así:

```
SELECT Nombre, Apellido FROM Clientes
```

Insertar datos

El comando para insertar datos en una tabla es INSERT. Deben respetar el orden de las columnas con los datos que se van a guardar y deben estar separados por comas las columnas y es posible ignorar los campos que permiten valores nulos.

```
INSERT INTO nombre_tabla (columna1, columna2, columna3, ...)
VALUES (valor1, valor2, valor3, ...);
```

Siguiendo con el ejemplo de la tabla Clientes, para agregar un nuevo cliente debemos ejecutar la siguiente instrucción:

```
INSERT INTO Clientes (Nombre, Apellido, Id_Localidad) VALUES ('Juan',
'Gomez', 15)
```

Actualizando registros

La instrucción UPDATE sirve para actualizar los registros existentes. Es necesario definir una condición que permita identificar los registros que se quieren actualizar:

```
UPDATE Nombre_tabla  
SET columna1 = valor1, columna2 = valor2, ...  
WHERE condicion;
```

Si queremos actualizar un registro de nuestra base de clientes para cambiar el nombre de un cliente sabiendo previamente cuál es el id del mismo, la sentencia sería la siguiente:

```
UPDATE Clientes SET Nombre = 'Juan Carlos', Apellido = 'Gómez' WHERE  
Id_Cliente = 10
```

Borrar datos

Para borrar registros necesitamos la instrucción DELETE. Nuevamente es necesario y muy importante tener una condición que nos permita identificar los registros a eliminar. En caso de que no se defina esta condición es posible que perdamos todos los registros de nuestra tabla o base de datos. La instrucción sería:

```
DELETE FROM Nombre_tabla WHERE condicion;
```

Si queremos borrar al cliente número 10 de nuestra tabla de clientes la instrucción nos quedaría:

```
DELETE FROM Clientes WHERE Id_Cliente = 10
```

Operadores

En ejemplos anteriores vimos la cláusula WHERE que nos sirve para definir condiciones. Es importante explorar las posibilidades que nos da esta cláusula, ya

que además de los operadores lógicos que ya conocemos (=,!=,<,>,<=, >=) tenemos otros como por ejemplo

- **IS NULL** para seleccionar aquellos campos que sean nulos
- **IS NOT NULL** para seleccionar aquellos campos que no sean nulos
- **IN** para seleccionar aquellos campos que estén dentro de una lista
- **LIKE** para buscar con algún patrón

También tenemos la posibilidad de unir condiciones bajo los operadores AND, OR y NOT :

```
SELECT columna1, columna2, ...  
FROM Nombre_tabla  
WHERE condicion1 AND condiciones2 OR condicion3 NOT condicion4 ...;
```

Ordenando datos

Ya vimos cómo seleccionar datos, pero seguramente nos interesaría que esa selección venga ordenada por algún campo y bajo algún criterio. Para esa acción tenemos la instrucción ORDER BY. Se pueden especificar una o más columnas. La sentencia es la siguiente:

```
SELECT columna1, columna2, ...  
FROM Nombre_tabla  
ORDER BY columna1, columna2, ... ASC|DESC;
```

Por ejemplo, si quisiéramos obtener todos los registros de la tabla Clientes ordenados por Apellido ascendente, nuestra consulta quedaría de la siguiente manera:

```
SELECT * FROM Clientes ORDER BY Apellido ASC
```

Si a la condición anterior le agregamos que ordene también por el campo Nombre nuestra consulta quedaría de la siguiente manera:

```
SELECT * FROM Clientes ORDER BY Apellido ASC, Nombre ASC
```

Funciones de Agregación

Una función de agregación es un cálculo sobre un conjunto de valores, devolviendo un sólo valor, excepto por la función COUNT. Estas funciones ignoran los valores NULL. Por lo general, debemos combinar las funciones de agregado con la cláusula GROUP BY. Algunas de ellas son MAX (Máximo), MIN (Mínimo), AVG (Promedio), SUM (Suma), COUNT (Cantidad) entre otras.

Por ejemplo, si tuviéramos una tabla Transferencias con un campo monto y quisiéramos obtener el monto máximo, nuestra consulta sería la siguiente: SELECT MAX(Monto) FROM Transferencias.

Subconsultas

Una subconsulta es una instrucción SELECT anidada dentro de otra instrucción SELECT, las cuales pueden contener condiciones como las mencionadas anteriormente. Para generar una subconsulta debemos tener presente ciertas reglas como: escribir la subconsulta entre paréntesis, especificar en ella sólo una columna (si no se utiliza IN, ANY, ALL o EXISTS), que no contenga la cláusula GROUP BY.

Además que no se puede recuperar datos de la misma tabla a la cual se realice la sentencia UPDATE o DELETE.

Por ejemplo, si quisiéramos obtener los Clientes que tienen en el campo Id_Localidad cualquier ID que esté en la tabla Provincias siendo Córdoba el nombre de la misma, nuestra consulta sería la siguiente:

```
SELECT * FROM Clientes WHERE Id_Localidad IN(SELECT Id_Localidad FROM Provincias WHERE Nombre = 'Córdoba')
```

La subconsulta obtendrá todos los valores de Id_Localidad que contenga la provincia Córdoba y la consulta obtendrá los Clientes que tengan el valor de Id_Localidad que estén en la lista de la subconsulta.

Conjuntos y JOINS en Bases de Datos

Hemos visto cómo seleccionar datos de una tabla sin embargo hasta el momento no tocamos el caso del cual hemos hablado tanto durante este capítulo: las tablas

relacionadas. Recordemos los conceptos de Clave Primaria y Clave Foránea porque ahora son esenciales ya que nos permiten realizar los JOIN de una forma mucho mas eficiente.

¿Qué son las consultas JOIN?

JOIN es el proceso de tomar datos de varias tablas y colocarlos en un mismo conjunto. Por tanto, una instrucción de “SQL JOIN” en un comando Select combina las columnas entre una o más tablas en una base de datos relacional y retorna a un conjunto de datos. Entonces, la cláusula JOIN nos permite asociar 2 o más tablas, en base a una columna que tengan en común.

Estas asociaciones tienen 3 formas y en cada caso se obtienen resultados diferentes. Es muy importante entender estos conceptos porque la tarea de hacer JOINS es muy común en las bases de datos.

A continuación, vamos a analizar la cláusula JOIN y sus variantes. Pero primero, empecemos definiendo 2 tablas, que serán la base para nuestros ejemplos posteriores.

Por un lado, vamos a tener una tabla Empleados que almacenará una lista de empleados y el id del departamento al que pertenecen. Prestemos atención a un detalle en la siguiente tabla: al caso de Williams que no tiene un departamento asignado (NULL).

Nombre	Departamentoid
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Y por otro lado, una tabla Departamentos con la lista de departamentos que existen en la empresa. En esta tabla también hay un detalle importante a tener en cuenta: El departamento de Marketing no tiene ningún empleado asociado:

Id	Nombre
31	Sales
33	Engineering
34	Clerical
35	Marketing

Continuemos con el análisis de los tipos de JOIN:

INNER JOIN

Esta es la opción predeterminada, también conocida como simplemente JOIN. Si no se especifica el tipo de unión, se establecerá de manera predeterminada como la unión interna. Esto implica que, si estamos uniendo dos tablas en una columna común, sólo retornará los datos que coincidan en ambas tablas.

La consulta genérica sería:

```
SELECT nombre_columna1, nombre_columna2, ...  
FROM Nombre_tabla1  
INNER JOIN Nombre_tabla2  
ON Nombre_tabla1.nombre_columna_clave = Nombre_tabla2.nombre_columna_clave;
```

Una breve explicación de esta consulta: para realizar el JOIN, en cualquiera de los siguientes casos, es necesario mencionar cuáles son las tablas que se van a asociar y qué campo se va a utilizar. El FROM indica la tabla de origen, luego del JOIN la tabla de destino y el ON cuál es la condición de igualdad para la asociación.

La consulta para este caso sería la siguiente:

```
SELECT * FROM Empleados E JOIN Departamentos D ON E.DepartamentoId = D.Id
```

Dando como resultado la siguiente tabla:

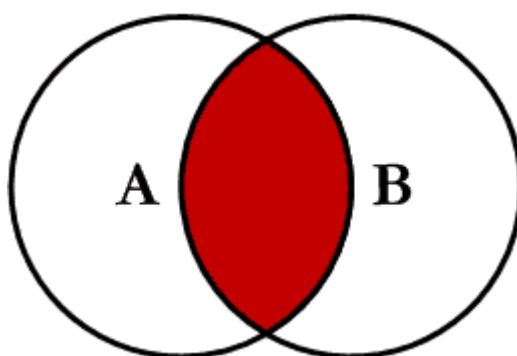
Nombre	Departmentold	Id	Nombre
Rafferty	31	31	Sales
Jones	33	33	Engineering
Heisenberg	33	33	Engineering
Robinson	34	34	Clerical
Smith	34	34	Clerical

A partir de la tabla anterior tenemos que notar lo siguiente:

- El empleado "Williams" no aparece en los resultados, ya que no pertenece a ningún departamento existente.
- El departamento "Marketing" tampoco aparece, ya que ningún empleado pertenece a dicho departamento.

¿Por qué ocurre esto? Porque como se dijo antes, JOIN muestra como resultado la intersección de ambas tablas.

Visto mediante un [diagramas de Venn](#), podemos observar que del conjunto A (Empleados) y del conjunto B (Departamentos), el INNER JOIN representa aquellos objetos que forman parte de la unión de ambos conjuntos. Lo que se simboliza con el espacio en rojo:



inner join. (n.d.).

También hay que tener en cuenta que, en los resultados vemos 4 columnas. Las 2 primeras se corresponden con la tabla Empleados y las últimas con Departamentos. Esto ocurre porque estamos seleccionando todas las columnas con un *. Si queremos, podemos ser específicos y seleccionar sólo 2 columnas, para eso debemos configurar nuestra consulta para seleccionar dichas columnas:

```
SELECT E.Nombre AS 'Empleado', D.Nombre AS 'Departamento' FROM Empleados E
JOIN Departamentos D ON E.DepartamentoId = D.Id
```

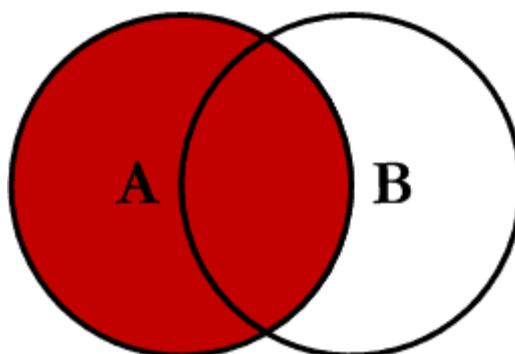
Que nos devolverá la siguiente tabla:

Empleado	Departamento
Rafferty	Sales
Jones	Engineering
Heisenberg	Engineering
Robinson	Clerical
Smith	Clerical

LEFT JOIN

A diferencia de un INNER JOIN, en donde se busca una intersección respetada por ambas tablas, con LEFT JOIN damos prioridad a la tabla de la izquierda, y buscamos en la tabla de la derecha. Si no existe ninguna coincidencia para alguna de las filas de la tabla de la izquierda, de igual forma todos los resultados de la primera tabla se muestran.

En este caso empezamos por el diagrama de Venn para luego ver la tabla, y comprender la lógica desde otro punto de vista. Nuevamente el conjunto A representa la tabla de empleados y el conjunto B la tabla de Departamentos:



LEFT JOIN. (n.d.).

Según el gráfico anterior buscamos obtener todos los elementos del conjunto de la IZQUIERDA (LEFT), es decir el A (Empleados) con la unión de aquellos elementos que coincidan con el conjunto B (Departamentos).

Entonces la tabla que vamos a obtener es la siguiente:

Empleado	Departamento
Rafferty	Sales
Jones	Engineering
Heisenberg	Engineering
Robinson	Clerical
Smith	Clerical
Williams	NULL

Y a partir de la tabla anterior tenemos que notar lo siguiente:

- El empleado "Williams" aparece en los resultados, ya que pertenece al conjunto A y el LEFT JOIN nos trae todos sus elementos, tengan o no una relación con el conjunto B

La consulta SQL para este LEFT JOIN sería:

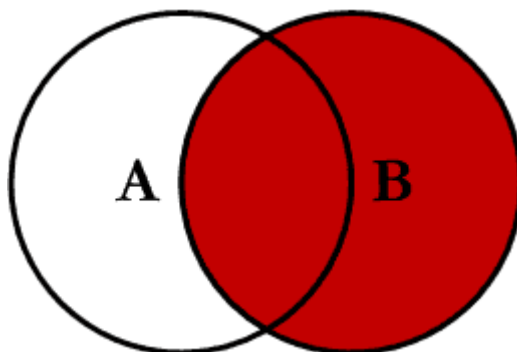
```
SELECT
  E.Nombre as 'Empleado',
  D.Nombre as 'Departamento'
FROM Empleados E
LEFT JOIN Departamentos D
ON E.DepartamentoId = D.Id
```

Hay que hacer algunas observaciones al código de la imagen anterior:

- La tabla Empleados es la primera tabla en aparecer en la consulta (en el FROM), por lo tanto, ésta es la tabla LEFT (izquierda), y todas sus filas se mostrarán en los resultados.
- La tabla Departamentos es la tabla de la derecha (aparece luego del LEFT JOIN). Por lo tanto, si se encuentran coincidencias, se mostrarán los valores correspondientes, pero si no, aparecerá NULL en los resultados.

RIGHT JOIN

En el caso de RIGHT JOIN la situación es muy similar, pero aquí se da prioridad a la tabla de la derecha. Como se ilustra a continuación con el diagrama de Venn:



RIGHT JOIN. (n.d.).

Entonces, nuevamente partiendo del gráfico de conjuntos tenemos que ver que el resultado será: todos los elementos del conjunto B (Departamentos) en unión con aquellos elementos que coincidan con el conjunto A (Empleados). El resultado es la siguiente tabla:

Empleado	Departamento
Rafferty	Sales
Jones	Engineering
Heisenberg	Engineering
Robinson	Clerical
Smith	Clerical
NULL	Marketing

Y a partir de la tabla anterior tenemos que notar lo siguiente:

- El departamento de Marketing, que no tiene ningún empleado asociado, aparece con el valor NULL. Como mencionamos antes, vamos a obtener todos los elementos del conjunto B (Departamentos), tengan o no una relación con A (Empleados)
- Del punto anterior también debemos mencionar que el resultado mostrará todos los departamentos al menos 1 vez.

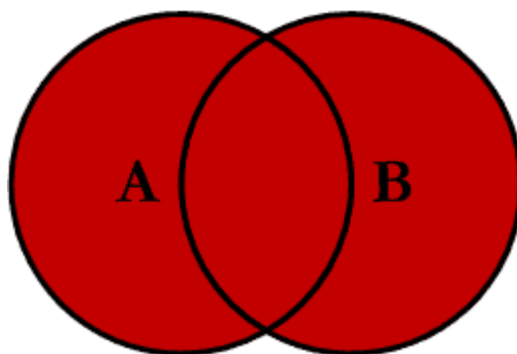
La consulta SQL para este RIGHT JOIN sería:

```
SELECT
  E.Nombre as 'Empleado',
  D.Nombre as 'Departamento'
FROM Empleados E
RIGHT JOIN Departamentos D
ON E.DepartamentoId = D.Id
```

FULL JOIN

Mientras que LEFT JOIN muestra todas las filas de la tabla izquierda, y RIGHT JOIN muestra todas las correspondientes a la tabla derecha, FULL OUTER JOIN (o simplemente FULL JOIN) se encarga de mostrar todas las filas de ambas tablas, sin importar que no existan coincidencias (usará NULL como un valor por defecto para dichos casos).

El diagrama de conjuntos será el siguiente:



Full JOIN. (n.d.).

En nuestra tabla de ejemplo se mostrará tanto el empleado "Williams" a pesar de que no está asignado a ningún departamento, y el departamento de "Marketing" a pesar que aún nadie está trabajando allí:

Empleado	Departamento
Rafferty	Sales
Jones	Engineering
Heisenberg	Engineering
Robinson	Clerical
Smith	Clerical
Williams	NULL
NULL	Marketing

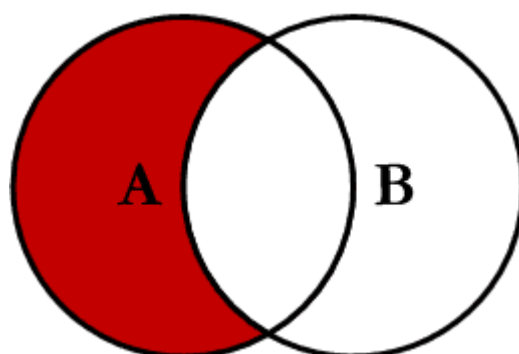
La consulta SQL para este FULL JOIN sería:

```
SELECT
  E.Nombre as 'Empleado',
  D.Nombre as 'Departamento'
FROM Empleados E
FULL JOIN Departamentos D
ON E.DepartamentoId = D.Id
```

Más variantes

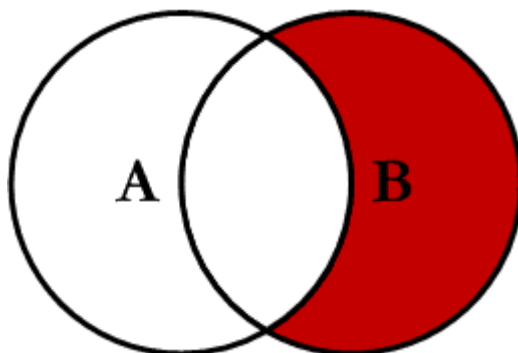
Si miramos con atención los siguientes diagramas de Venn, vamos a notar que es posible formar incluso más combinaciones, al momento de seleccionar datos.

Por ejemplo, tenemos el siguiente caso, conocido como LEFT EXCLUDING JOIN:



LEFT EXCLUDING JOIN. (n.d.).

Y de igual manera RIGHT EXCLUDING JOIN:

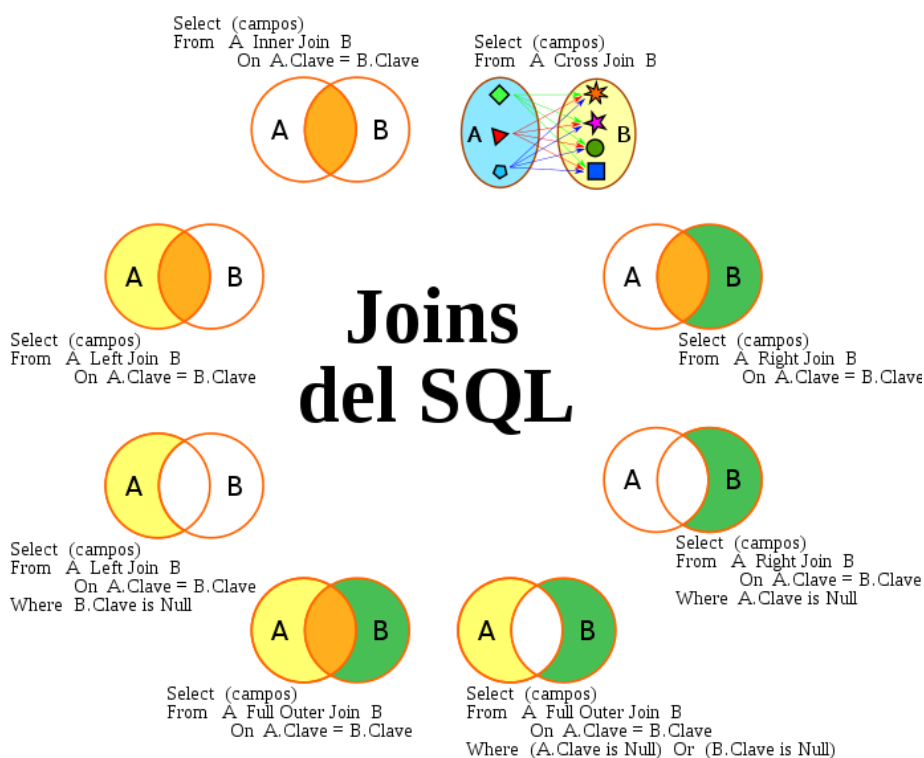


RIGHT EXCLUDING JOIN. (n.d.).

Estas combinaciones son posibles de lograr si añadimos algunas condiciones a nuestras consultas, haciendo uso de la cláusula WHERE. Por ejemplo, siguiendo el ejemplo que estamos viendo (donde Empleados es la tabla izquierda y Departamentos la tabla derecha):

1. LEFT EXCLUDING JOIN nos permitirá obtener la lista de empleados que aún no han sido asignados a ningún departamento de trabajo.
2. Mientras que RIGHT EXCLUDING JOIN nos mostrará la lista de departamentos que no tienen a ningún trabajador asociado.

Te dejamos a continuación una imagen que sintetiza todos los tipos de JOINS con sus respectivas consultas:



Transacciones en Base de Datos

¿Qué son las transacciones?

Las transacciones en SQL son unidades o secuencias de trabajo realizadas de forma ordenada y separada en una base de datos. Normalmente representan cualquier cambio en la base de datos, y tienen dos objetivos principales:

1. Proporcionar secuencias de trabajo fiables que permitan poder recuperarse fácilmente ante errores y mantener una base de datos consistente incluso frente a fallos del sistema.
2. Proporcionar aislamiento entre programas accediendo a la vez a la base de datos.

Una transacción es una propagación de uno o más cambios en la base de datos, ya sea cuando se crea, se modifica o se elimina un registro. En la práctica suele consistir en la agrupación de consultas SQL y su ejecución como parte de una transacción.

Propiedades de las transacciones

Las transacciones siguen cuatro propiedades básicas, bajo el acrónimo ACID (Atomicity, Consistency, Isolation, Durability):

1. **Atomicidad:** aseguran que todas las operaciones dentro de la secuencia de trabajo se completen satisfactoriamente. Si no es así, la transacción se abandona en el punto del error y las operaciones previas retroceden a su estado inicial.
2. **Consistencia:** aseguran que la base de datos cambie estados en una transacción exitosa.
3. **Aislamiento:** permiten que las operaciones sean aisladas y transparentes unas de otras.
4. **Durabilidad:** aseguran que el resultado o efecto de una transacción completada permanezca en caso de error del sistema.

Control de las transacciones

Existen tres comandos básicos de control en las transacciones SQL:

- **COMMIT:** Para guardar los cambios.
- **ROLLBACK:** Para abandonar la transacción y deshacer los cambios que se hubieran hecho en la transacción.
- **SAVEPOINT:** Crea checkpoints, puntos concretos en la transacción donde poder deshacer la transacción hasta esos puntos.

Los comandos de control de transacciones se usan sólo con INSERT, DELETE y UPDATE. No pueden utilizarse creando tablas o borrando su contenido porque las operaciones se guardan automáticamente en la base de datos.

Ejemplo:

Un ejemplo habitual de transacción es el traspaso de una cantidad de dinero entre cuentas bancarias. Normalmente se realiza mediante dos operaciones distintas, una en la que se decrementa el saldo de la cuenta origen y otra en la que se incrementa el saldo de la cuenta destino. Para garantizar la integridad del sistema (es decir, para que no aparezca o desaparezca dinero) las dos operaciones deben ser atómicas, es decir que el sistema debe garantizar que, bajo cualquier circunstancia (incluso una caída del sistema), el resultado final es: o bien se han realizado las dos operaciones, o bien no se ha realizado ninguna.

```
DECLARE @Monto DECIMAL(18,2),
@CuentaADecrementar VARCHAR(12),
@CuentaAIncrementar VARCHAR(12)

/* Asignamos el monto de la transacción y las cuentas a afectar*/

SET @Monto = 1900
SET @CuentaADecrementar = '20161206'
SET @CuentaAIncrementar = '20161207'

BEGIN TRANSACTION
BEGIN TRY

/* Descontamos el monto de la cuenta a decrementar */
UPDATE CUENTAS SET SALDO = SALDO - @Monto WHERE NUMCUENTA =
@CuentaADecrementar

/* Registramos el movimiento */
INSERT INTO MOVIMIENTOS (IDCUENTA, SALDO_ANTERIOR, SALDO_POSTERIOR, IMPORTE,
FXMOVIMIENTO)

/* Incrementamos el monto de la cuenta a incrementar */
UPDATE CUENTAS SET SALDO = SALDO + @Monto WHERE NUMCUENTA =
@CuentaAIncrementar

/* Registramos el movimiento */
INSERT INTO MOVIMIENTOS (IDCUENTA, SALDO_ANTERIOR, SALDO_POSTERIOR, IMPORTE,
FXMOVIMIENTO)

/* Confirmamos la transaccion*/
COMMIT TRANSACTION

END TRY

BEGIN CATCH

/* Ocurrió un error, deshacemos los cambios*/
ROLLBACK TRANSACTION
PRINT 'Ha ocurrido un error!'

END CATCH
```

Bloqueo o Locking en Bases de Datos

El término real es “Locking” que del inglés lo traducimos como “Bloqueo”, hace referencia a la protección de información sobre objetos (datos), en los que consideramos diferentes transacciones u operaciones en una Base de Datos.

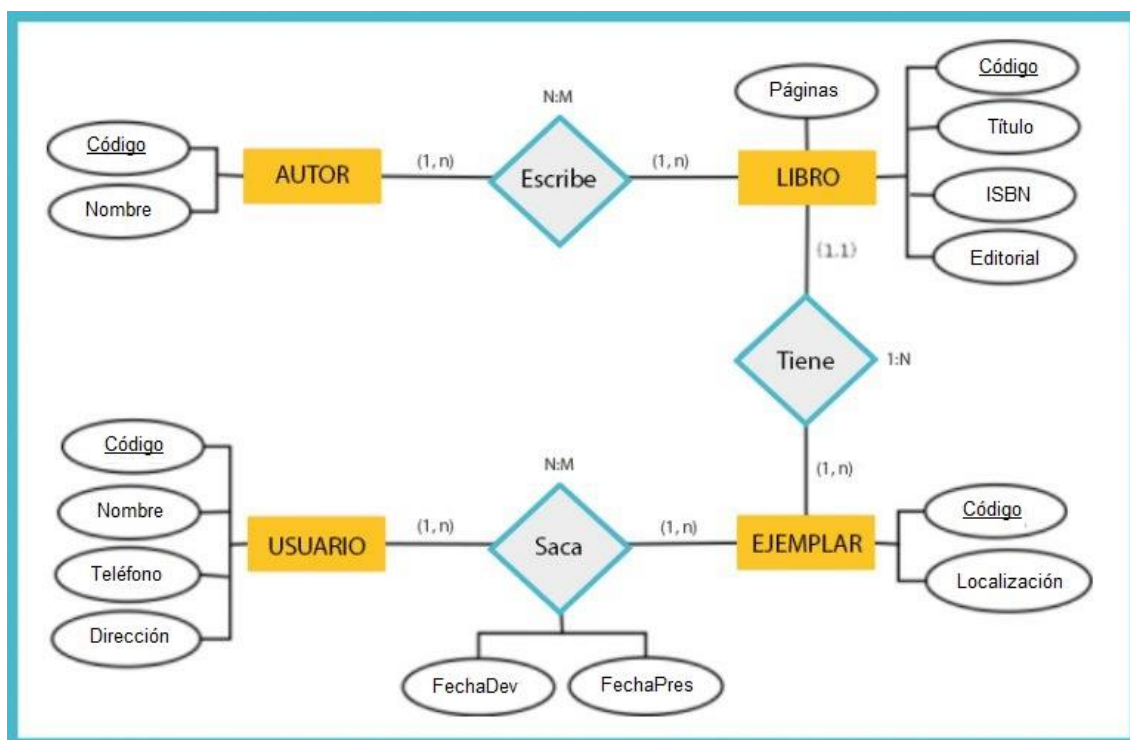
Los bloqueos son parte esencial del requisito de aislamiento de los datos y se utilizan para bloquear los datos afectados dentro de una operación. En el momento en que los datos están bloqueados, no se permitirá que otras operaciones lleguen a realizarse haciendo cambios en los datos que se guardan en los objetos afectados por el bloqueo que se realiza.

Una vez liberado el bloqueo, al confirmar los cambios o al revertir los cambios al estado inicial, se podrán realizar otras operaciones que permitirán hacer los cambios de datos que se requieran.

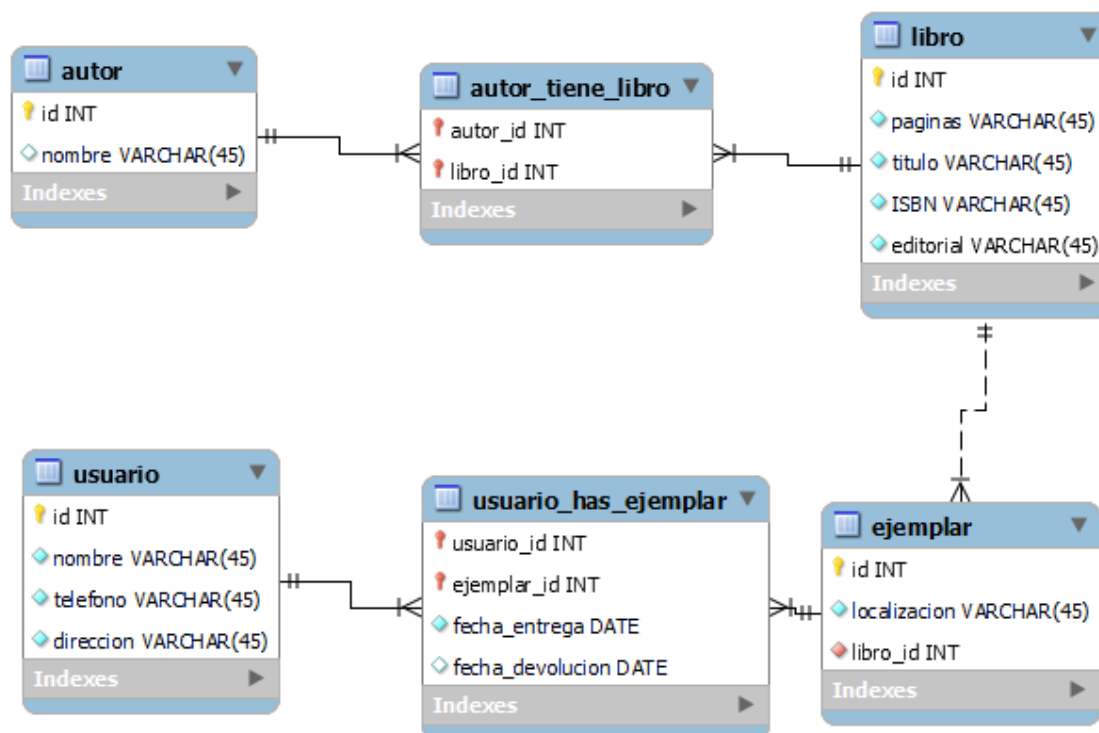
Esto, traducido al lenguaje de SQL, significa que cuando se realiza una operación, se antepone el bloqueo en un objeto, el resto de las operaciones que se requieren para dar el acceso al objeto serán forzadas a esperar hasta la liberación del bloqueo y que esa espera se registre con el tipo de espera correcto.

Diagrama Entidad Relación (DER)

En el siguiente video veremos cómo pasar de un DER al esquema lógico y finalmente a la Base de Datos. Partiremos de un DER que esquematiza el requerimiento de una pequeña biblioteca de barrio. Para este video necesitaremos tener el Workbench y MySQL instalados. Partiremos del siguiente DER:



Para llegar al siguiente esquema lógico:



Ya con nuestra base de datos creada, podemos empezar a completar algunos datos y practicar las consultas que ya vimos

Una vez que tenemos nuestra Base de Datos, es normal que los proyectos evolucionen y el cliente o usuarios del sistema pidan agregar nuevas funciones. ¿Qué pasaría si se nos pide agregar fotos a los libros? El frontend dirá, no hay problema ponemos un formulario y que suban las fotos. Luego el backend dirá que, si recibe las fotos desde la web, las guarda en una carpeta y se las manda a la base de datos Entonces nos llaman a nosotros y nos piden que hagamos el ajuste en la base de datos, tomate unos minutos para reflexionar la situación.