Base de datos

- Que es una base de datos
- Que es Mysql
- Que es SQL
- Modelo entidad relación
- Tipos de datos
- Componentes SQL
- Trabajar con varias tablas

¿Qué es una Base de datos?

Una base de datos es una colección organizada de información estructurada, o datos, típicamente almacenados en un sistema de computadora. Una base de datos es usualmente controlada por un sistema de gestión de base de datos (DBMS). En conjunto, los datos y el DBMS, junto con las aplicaciones que están asociados con ellos, se conocen como un sistema de base de datos, que a menudo se reducen a solo base de datos.

Ejemplos de base de datos

Algunos ejemplos posibles de bases de datos a lo largo de la historia son:

- **Guías telefónicas.** Aunque en desuso, estos voluminosos libros solían contener miles de números telefónicos asignados a hogares, empresas y particulares, para permitir al usuario dar con el que necesitaba. Eran engorrosos, pesados, pero completos.
- **Archivos personales.** El conjunto de los escritos de vida de un autor, investigador o intelectual a menudo son preservados en un archivo, que se organiza en base a la preservación y reproducción de los originales, permitiendo su consulta sin poner en riesgo el documento original.
- **Bibliotecas públicas.** El perfecto ejemplo de bases de datos, pues contienen miles o cientos de miles de registros pertenecientes a cada título de libro disponible para su préstamo, ya sea en sala o circulante, y del que puede haber más de un mismo ejemplar en el depósito. Los bibliotecólogos se encargan de diseñar estos sistemas y velar por su funcionamiento.
- **Registros de transacciones.** Las operaciones realizadas con una tarjeta de crédito, así como las llamadas realizadas con un celular, u otro tipo de transacciones comerciales cotidianas, generan todo un conjunto de registros que van a dar a una base de datos de la empresa.
- **Historial médico.** Cada vez que acudimos al doctor o a un hospital, se actualiza la información respecto a nuestra salud, al tratamiento recibido y demás detalles médicos en un archivo que lleva registro de nuestra historia médica, en caso de que a futuro se requiera conocer datos específicos, como operaciones o tratamientos recibidos.

Los datos dentro de los tipos más comunes de bases de datos en funcionamiento hoy en día se modelan típicamente en filas y columnas en una serie de tablas para que el procesamiento y la consulta de datos sean eficientes. Luego se puede acceder, administrar, modificar, actualizar, controlar y organizar fácilmente los datos. La mayoría de las bases de datos utilizan lenguaje de consulta estructurado (SQL) para escribir y consultar datos.

Ventajas de las bases de datos

Control sobre la redundancia de datos:

Los sistemas de ficheros almacenan varias copias de los mismos datos en ficheros distintos. Esto hace que se desperdicie espacio de almacenamiento, además de provocar la falta de consistencia de datos.

En los sistemas de bases de datos todos estos ficheros están integrados, por lo que no se almacenan varias copias de los mismos datos. Sin embargo, en una base de datos no se puede eliminar la redundancia completamente, ya que en ocasiones es necesaria para modelar las relaciones entre los datos.

Consistencia de datos:

Eliminando o controlando las redundancias de datos se reduce en gran medida el riesgo de que haya inconsistencias. Si un dato está almacenado una sola vez, cualquier actualización se debe realizar sólo una vez, y está disponible para todos los usuarios inmediatamente. Si un dato está duplicado y el sistema conoce esta redundancia, el propio sistema puede encargarse de garantizar que todas las copias se mantienen consistentes.

Compartir datos:

En los sistemas de ficheros, los ficheros pertenecen a las personas o a los departamentos que los utilizan. Pero en los sistemas de bases de datos, la base de datos pertenece a la empresa y puede ser compartida por todos los usuarios que estén autorizados.

Mantenimiento de estándares:

Gracias a la integración es más fácil respetar los estándares necesarios, tanto los establecidos a nivel de la empresa como los nacionales e internacionales. Estos estándares pueden establecerse sobre el formato de los datos para facilitar su intercambio, pueden ser estándares de documentación, procedimientos de actualización y también reglas de acceso.

Mejora en la integridad de datos:

La integridad de la base de datos se refiere a la validez y la consistencia de los datos almacenados. Normalmente, la integridad se expresa mediante restricciones o reglas que no se pueden violar. Estas restricciones se pueden aplicar tanto a los datos, como a sus relaciones, y es el SGBD quien se debe encargar de mantenerlas.

Mejora en la seguridad:

La seguridad de la base de datos es la protección de la base de datos frente a usuarios no autorizados. Sin unas buenas medidas de seguridad, la integración de datos en los sistemas de bases de datos hace que éstos sean más vulnerables que en los sistemas de ficheros.

Mejora en la accesibilidad a los datos:

Muchos SGBD proporcionan lenguajes de consultas o generadores de informes que permiten al usuario hacer cualquier tipo de consulta sobre los datos, sin que sea necesario que un programador escriba una aplicación que realice tal tarea.

Mejora en la productividad:

El SGBD proporciona muchas de las funciones estándar que el programador necesita escribir en un sistema de ficheros. A nivel básico, el SGBD proporciona todas las rutinas de manejo de ficheros típicas de los programas de aplicación.

El hecho de disponer de estas funciones permite al programador centrarse mejor en la función específica requerida por los usuarios, sin tener que preocuparse de los detalles de implementación de bajo nivel.

Mejora en el mantenimiento:

En los sistemas de ficheros, las descripciones de los datos se encuentran inmersas en los programas de aplicación que los manejan.

Esto hace que los programas sean dependientes de los datos, de modo que un cambio en su estructura, o un cambio en el modo en que se almacena en disco, requiere cambios importantes en los programas cuyos datos se ven afectados.

Sin embargo, los SGBD separan las descripciones de los datos de las aplicaciones. Esto es lo que se conoce como independencia de datos, gracias a la cual se simplifica el mantenimiento de las aplicaciones que acceden a la base de datos.

Aumento de la concurrencia:

En algunos sistemas de ficheros, si hay varios usuarios que pueden acceder simultáneamente a un mismo fichero, es posible que el acceso interfiera entre ellos de modo que se pierda información o se pierda la integridad. La mayoría de los SGBD gestionan el acceso concurrente a la base de datos y garantizan que no ocurran problemas de este tipo.

Mejora en los servicios de copias de seguridad:

Muchos sistemas de ficheros dejan que sea el usuario quien proporcione las medidas necesarias para proteger los datos ante fallos en el sistema o en las aplicaciones. Los usuarios tienen que hacer copias de seguridad cada día, y si se produce algún fallo, utilizar estas copias para restaurarlos.

En este caso, todo el trabajo realizado sobre los datos desde que se hizo la última copia de seguridad se pierde y se tiene que volver a realizar. Sin embargo, los SGBD actuales funcionan de modo que se minimiza la cantidad de trabajo perdido cuando se produce un fallo.

Desventajas de las bases de datos

Complejidad:

Los SGBD son conjuntos de programas que pueden llegar a ser complejos con una gran funcionalidad. Es preciso comprender muy bien esta funcionalidad para poder realizar un buen uso de ellos.

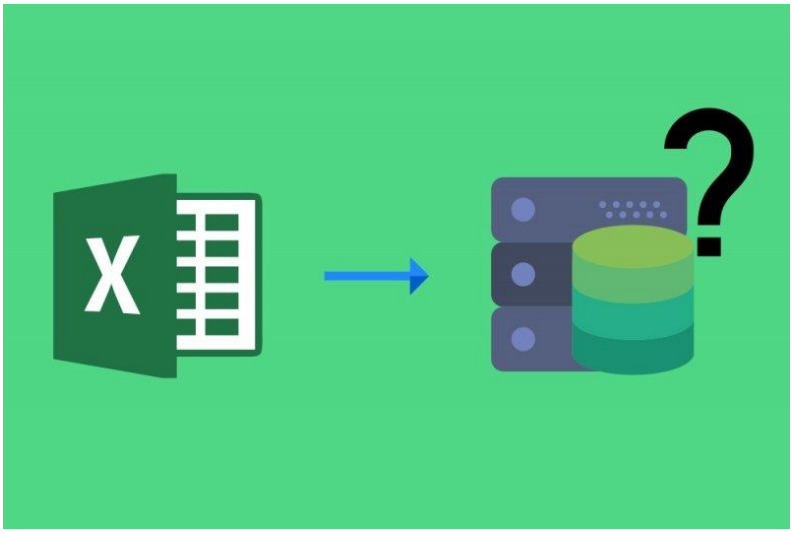
Coste del equipamiento adicional:

Tanto el SGBD, como la propia base de datos, pueden hacer que sea necesario adquirir más espacio de almacenamiento. Además, para alcanzar las prestaciones deseadas, es posible que sea necesario adquirir una máquina más grande o una máquina que se dedique solamente al SGBD. Todo esto hará que la implantación de un sistema de bases de datos sea más cara.

Vulnerable a los fallos:

El hecho de que todo esté centralizado en el SGBD hace que el sistema sea más vulnerable ante los fallos que puedan producirse. Es por ello que deben tenerse copias de seguridad (Backup).

¿Cuál es la diferencia entre una base de datos y una hoja de cálculo?



Las bases de datos y las hojas de cálculo (como Microsoft Excel) son dos formas convenientes de almacenar información. Las principales diferencias entre las dos son:

-Cómo se almacenan y manipulan los datos

-Quién puede acceder a los datos

-Cuántos datos se pueden almacenar

Las hojas de cálculo se diseñaron originalmente para un usuario, y sus características lo reflejan. Son muy buenas para un solo usuario o un pequeño número de usuarios que no necesitan manipular una gran cantidad de datos complicados. Las bases de datos, por otro lado, están diseñadas para contener colecciones mucho más grandes de información organizada, cantidades masivas en ocasiones. Las bases de datos permiten a múltiples usuarios al mismo tiempo acceder y consultar los datos de forma rápida y segura utilizando una lógica y un lenguaje altamente complejos.

Tipos de bases de datos

Hay muchos tipos diferentes de bases de datos. La mejor base de datos para una organización específica depende de cómo la organización pretende utilizar los datos.

Bases de datos relacionales. Las bases de datos relacionales se popularizaron en los años ochenta. Los elementos de una base de datos relacional se organizan como un conjunto de tablas con columnas y filas. La tecnología de base de datos relacional proporciona la manera más eficiente y flexible de acceder a información estructurada.

Bases de datos orientadas a objetos. La información en una base de datos orientada a objetos se representa en forma de objetos, como en la programación orientada a objetos.

Bases de datos distribuidas. Una base de datos distribuida consta de dos o más archivos ubicados en diferentes sitios. La base de datos puede almacenarse en múltiples computadoras, ubicadas en la misma ubicación física o dispersas en diferentes redes.

Almacenes de datos. Un almacén de datos es un tipo de base de datos diseñada específicamente para consultas y análisis rápidos, y funciona como un depósito central de datos.

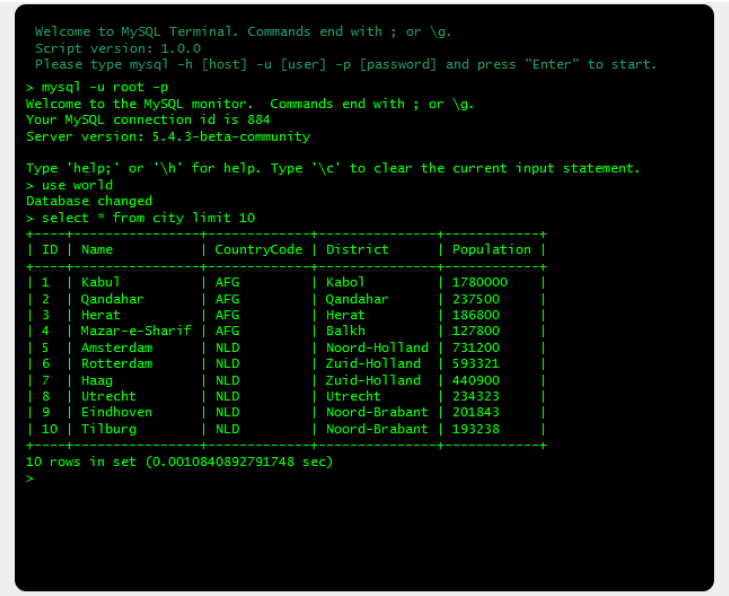
Bases de datos NoSQL. Una NoSQL, o una base de datos no relacional, permite que los datos no estructurados y semiestructurados se almacenen y manipulen, a diferencia de una base de datos relacional, que define cómo deben componerse todos los datos insertados en la base de datos. Las bases de datos NoSQL se hicieron populares a medida que las aplicaciones web se hacían más comunes y más complejas.

Bases de datos orientadas a grafos. Una base de datos orientada a grafos almacena datos en términos de entidades y las relaciones entre entidades.

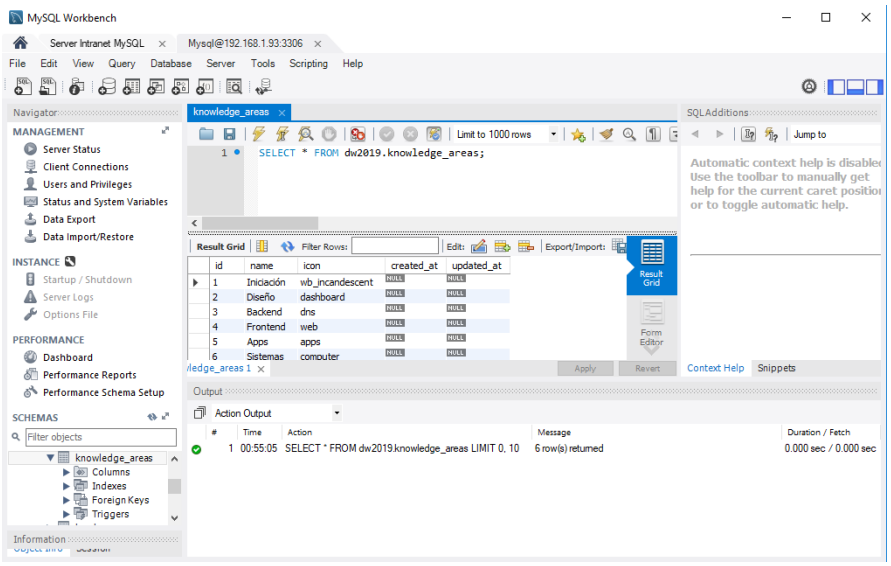
Bases de datos OLTP. Una base de datos OLTP es una base de datos analítica y rápida diseñada para un gran número de transacciones realizadas por múltiples usuarios.

Tipos de Clientes de Base de datos

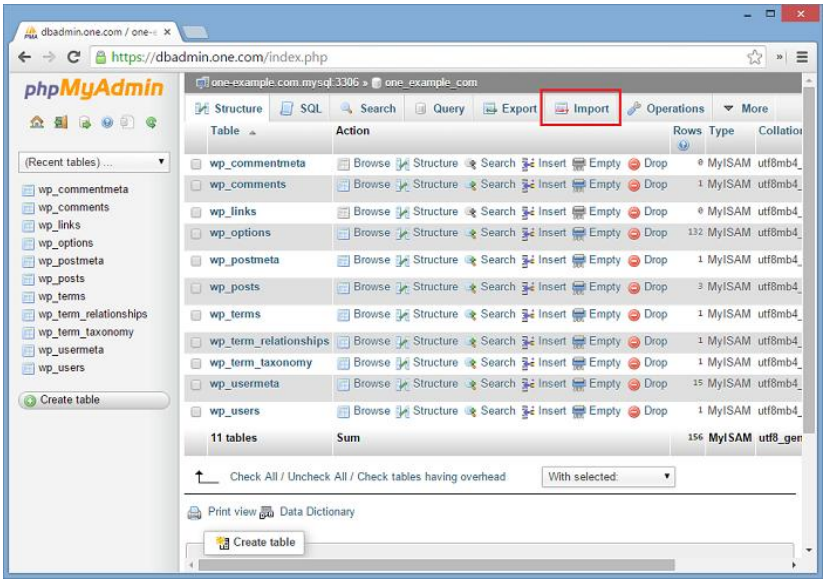
Cliente CLI (Command Line Interface): Es un cliente que interactúa con la base de datos mediante el uso de la consola:



Cliente GUI (graphicaluser interface): Es un cliente que interactúa con la base de datos mediante el uso de una aplicación gráfica.



Cliente Web: Es un cliente que interactúa con la base de datos a través de una página web mediante el uso de un navegador



¿Qué es el software de base de datos?

El software de base de datos se utiliza para crear, editar y mantener archivos y registros de bases de datos, lo que facilita la creación de archivos y registros, la entrada de datos, la edición de datos, las actualizaciones y los informes. El software también se encarga del almacenamiento de datos, las copias de seguridad y los informes, el control de acceso múltiple y la seguridad. La sólida seguridad de las bases de datos es especialmente importante hoy en día, ya que el robo de información se vuelve más frecuente. En ocasiones, el software de base de datos también se denomina "sistema de administración de bases de datos" (DBMS).

El software de base de datos simplifica la gestión de la información al permitirles a los usuarios almacenar datos en una forma estructurada y luego, acceder a ellos. Por lo general, tiene una

interfaz gráfica para ayudar a crear y administrar los datos y, en algunos casos, los usuarios pueden crear sus propias bases de datos mediante el software de base de datos.

¿Qué es un sistema de administración de base de datos (DBMS)?

Una base de datos generalmente requiere un programa completo de software de base de datos, que se conoce como sistema de administración de bases de datos (DBMS). Un DBMS sirve como una interfaz entre la base de datos y sus usuarios o programas finales, lo que permite a los usuarios recuperar, actualizar y administrar cómo se organiza y optimiza la información. Un DBMS también facilita la supervisión y el control de las bases de datos, lo que permite una variedad de operaciones administrativas, como la supervisión del rendimiento, el ajuste, las copias de seguridad y la recuperación.

Algunos ejemplos de software de bases de datos o DBMS populares incluyen MySQL, Microsoft Access, Microsoft SQL Server, FileMaker Pro, Oracle Database y dBASE.

¿Qué es MySQL?

MySQL es un sistema de gestión de bases de datos relacionales de código abierto basado en SQL. Fue diseñado y optimizado para aplicaciones web y puede ejecutarse en cualquier plataforma. A medida que surgían nuevos y diferentes requisitos con Internet, MySQL se convirtió en la plataforma elegida por los desarrolladores web y las aplicaciones basadas en la web. Debido a que está diseñada para procesar millones de consultas y miles de transacciones, MySQL es una opción popular para las empresas de comercio electrónico que necesitan administrar múltiples transferencias de dinero. La flexibilidad bajo demanda es la característica principal de MySQL.

MySQL es el DBMS detrás de algunos de los mejores sitios web y aplicaciones basadas en la web del mundo, incluyendo Airbnb, Uber, LinkedIn, Facebook, Twitter y YouTube.

Uso de bases de datos para mejorar el rendimiento empresarial y la toma de decisiones

Con la recopilación masiva de datos del Internet of Things, que transforma la vida y la industria en todo el mundo, las empresas hoy en día tienen acceso a más datos que nunca. Las organizaciones con visión de futuro ahora pueden usar bases de datos para ir más allá del almacenamiento de datos básicos y las transacciones para analizar grandes cantidades de datos de múltiples sistemas. Mediante el uso de bases de datos y otras herramientas informáticas y de inteligencia empresarial, las organizaciones ahora pueden aprovechar los datos que recopilan para ejecutar de manera más eficiente, permitir una mejor toma de decisiones y volverse más ágiles y escalables.

La base de datos independiente está lista para proporcionar un impulso significativo a estas capacidades. Debido a que las bases de datos independientes automatizan procesos manuales costosos y prolongados, liberan a los usuarios comerciales para que sean más proactivos con sus datos. Al tener un control directo sobre la capacidad de crear y usar bases de datos, los usuarios obtienen control y autonomía mientras mantienen importantes estándares de seguridad.

Características

Veamos cuáles son algunas de las principales características de MySQL que han hecho que se convierta en uno de los sistemas de bases de datos que más se utilizan en todo el mundo.

Es gratuito: sin dudas un punto importantísimo. Si bien MySQL al día de hoy es un software propiedad de Oracle, esta empresa aún tiene disponible para el público general la versión

gratuita de MySQL, la cual puede ser descargada por quien desee hacerlo. También se ofrece una versión de pago, aunque el uso de la misma es extremadamente reducido.

Es multiplataforma: otra gran característica de MySQL que ha permitido que sea tan popular es su compatibilidad con muchos sistemas operativos, ya que funciona perfectamente en Solaris, FreeBSD, Linux, Mac OS y Windows, así que prácticamente cualquier computadora puede correrlo.

Interfaz gráfica: si bien MySQL puede ser usado mediante una consola en cualquier sistema operativo, a menudo se prefiere darle uso por medio de una interfaz gráfica, ya que las mismas suelen facilitar diversas tareas, ya que dadas las características de las bases de datos no siempre es sencillo operar en ellas. En el caso particular de MySQL se dispone de excelentes interfaces como por ejemplo MySQLWorkbench, phpMyAdminentre varios otros.

Motores: MySQL es compatible con casi una docena de motores de almacenamiento, sin embargo, por lo general la mayoría de las personas solo usan dos, los cuales son InnoDB y MyISAM.

Respaldos: por defecto MySQL incluye un sistema de respaldo de bases de datos que se conoce como mysqldump. Esta herramienta es utilizada para crear una copia comprimida de una base de datos, y posteriormente dicha copia puede ser utilizada para volver a crear la base de datos en cuestión en caso de que la misma tenga algún problema, pérdida de datos, etc.

Arquitectura multihilo: gracias a su arquitectura multihilo, MySQL nos brinda la posibilidad de que varios usuarios puedan acceder a los datos almacenados de forma simultánea. Si esto no fuera posible no hay duda que este gestor no sería ni la décima parte de lo popular que es hoy en día.

Privilegios: para acceder a las bases de datos y escribir, ver, borrar o modificar el contenido de las mismas se requiere de ciertos privilegios. En MySQL podemos crear usuarios y darle los privilegios que deseemos según las tareas que los mismos necesiten ejecutar dentro de las bases de datos.

¿Qué es el lenguaje de consulta estructurado (SQL)?

SQL (Structured Query Language)es un lenguaje de programación usado por casi todas las bases de datos relacionales para consultar, manipular y definir datos, y para proporcionar control de acceso. SQL se desarrolló por primera vez en IBM en la década de 1970 con Oracle como uno de los principales contribuyentes, lo que llevó a la implementación del estándar ANSI de SQL. SQL ha generado muchas extensiones por parte de compañías como IBM, Oracle y Microsoft. Aunque SQL todavía se usa ampliamente en la actualidad, comienzan a aparecer nuevos lenguajes de programación.

SQL es utilizado habitualmente no solo por los administradores de bases de datos, sino también por los desarrolladores que escriben scripts de integración de datos y por los analistas de datos que desean configurar y ejecutar consultas analíticas.

Los usos de SQL incluyen la modificación de las estructuras de tablas e índices de las bases de datos; la adición, actualización y eliminación de filas de datos; y la recuperación de subconjuntos de información de una base de datos para el procesamiento de transacciones y aplicaciones. Las consultas y otras operaciones SQL adoptan la forma de comandos escritos en forma de sentencias: las sentencias SQL más utilizadas son select, add, insert, update, delete, create, alter y truncate.

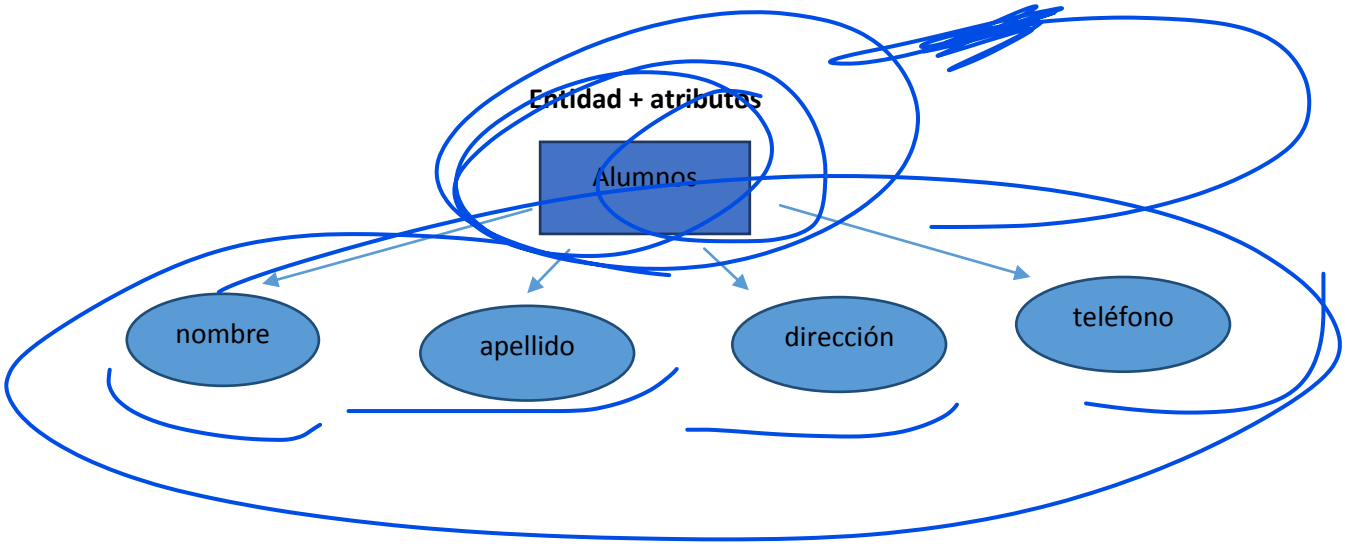
Modelo entidad relación

El modelo entidad relación es un concepto para diseñar Base de datos que posteriormente serán implementadas a través de un SGBD (MySql). Este modelo se representa a través de diagramas y está formado por varios elementos, de los cuales el principal es la entidad. El tipo de diagrama utilizado para realizar el modelado Entidad-Relación es el DER (Diagrama Entidad Relación), el cual pertenece al lenguaje de modelado universal (UML: Universal Modeling Languaje)

Se define una entidad (o instancia) como una unidad de una base de datos que contiene información. Esta unidad es una representación dentro de la base de datos de un objeto, persona, empresa... etc, que puede ser real o abstracto, y como tal posee ciertos atributos que la diferencian del resto de entidades. Así, por ejemplo, en una base de datos de un establecimiento educativo, una entidad podría ser los alumnos, otra las materias, otra los profesores. Cada una de estas entidades tendría ciertos atributos propios. Así, los alumnos tendrían atributos como nombre, edad, apellido.

Entidades

Los atributos definen o identifican las características propias y por lo general únicas de una entidad

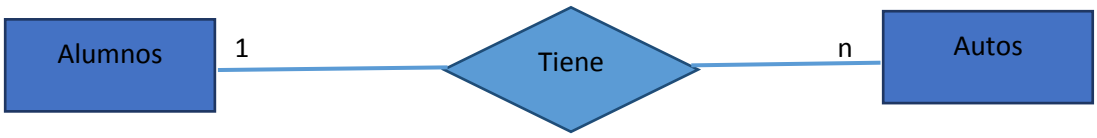


En una base de datos compleja pueden existir entidades relacionadas entre sí por diversos parámetros o atributos, de tal modo que la existencia de una puede ir ligada a la existencia de otra. Así, las entidades pueden ser fuertes (existen por si mismas) o débiles (su existencia depende de que exista otra entidad). Las relaciones entre entidades suelen describirse en el esquema de la estructura de la base de datos e incluso pueden agruparse entre sí para formar conjuntos de entidades.

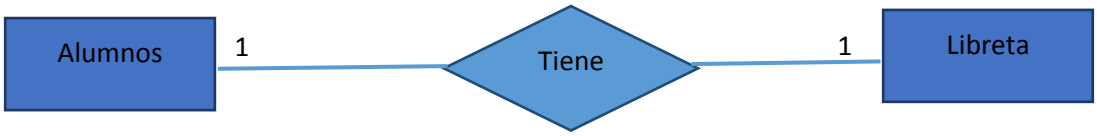
El modelo Entidad-Relación es de hecho uno de los más importantes a la hora de diseñar e implementar una base de datos con éxito. Mediante este modelo se relacionan una o varias entidades por sus atributos, que pueden ser comunes o no a varias de ellas.

Las relaciones tienen una característica denominada cardinalidad, la cual indica el sentido y la cantidad de "relaciones" existentes entre una entidad y otra. Estas pueden ser de los siguientes tipos:

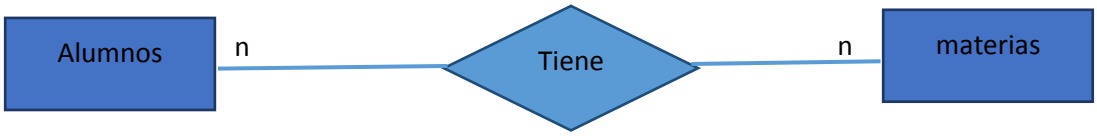
Relación uno a muchos (1-n): En el siguiente ejemplo se ve que un alumno puede tener muchos autos (n) y a su vez muchos autos pueden ser de un alumno.



Relación uno a uno (1-1): En el siguiente ejemplo se ve que un alumno puede tener una sola libreta, y esa libreta puede pertenecer a ese único alumno.



Relación muchos a muchos (n-n): En el siguiente ejemplo se ve que un alumno puede estar asignada a muchas materias, y las materias pueden tener muchos alumnos.



El siguiente diagrama permite entender de forma rápida y sencilla todas las tablas de nuestra base de datos.

Diagrama Físico (tablas, campos y registros)

Una tabla es una colección de datos con la misma estructura. Si se disponen los datos en una columna, esa columna siempre tiene el mismo tipo de datos, por ejemplo, un número de documento, un nombre, etc.

Cada **tabla** consta, si vamos a relacionarlo con el tema anterior podemos decir que una tabla es una entidad, de un cierto número de campos, en cada uno de los cuales se guarda un dato, que puede ser numérico, alfanumérico, fecha, etc.

Un campo es cada una de las columnas de la tabla y el nombre de la columna es el nombre del campo, relacionándolo con el tema anterior, un campo es un atributo.

Un registro es cada una de las filas de la tabla, y está formado por el dato de cada uno de los campos almacenados en una misma operación.

Nro_Emp	Nombre	Puesto	CP	Nacimiento	SAL	COMM	ID_DEPTO
7839	KING	PRESIDENT		17-NOV-81	5000		10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Tipos de datos en Mysql

Cada vez que tengamos que crear una tabla que sirva para almacenar datos de una aplicación Web, debemos poner a prueba nuestra capacidad para definir los tipos de datos que con mayor eficiencia puedan almacenar cada dato que necesitemos guardar.

Los campos de las tablas MySQL nos dan la posibilidad de elegir entre distintos tipos de datos:

-Numéricos

-Alfanuméricos

-Fechas y horas

Puede ser muy obvio poder distinguir a cuál de los tres grupos corresponderá, por ejemplo, un campo que guarde la "edad" de una persona.

Pero dentro de los distintos tipos de datos numéricos, ¿Cuál será la mejor opción?

Un número entero, pero, ¿Cuál de los distintos tipos de enteros disponibles?

¿Qué tipo de dato permitirá consumir menor espacio físico de almacenamiento y brindará la posibilidad de almacenar la cantidad de datos que se espera almacenar en ese campo? (dos dígitos, o máximo tres, en el caso de la edad).

Esas son preguntas que sólo podremos responder a partir del conocimiento de los distintos tipos de datos.

Datos numéricos

La diferencia entre uno y otro tipo de dato es simplemente el rango de valores que puede contener.

Dentro de los datos numéricos, podemos distinguir dos grandes ramas: enteros y decimales.

NUMÉRICOS ENTEROS

Comencemos por conocer las opciones que tenemos para almacenar datos que sean numéricos enteros (edades, cantidades, magnitudes sin decimales); poseemos una variedad de opciones:

Tipos de datos	Bytes	Valor mínimo	Valor máximo
TINYINT	1	-128	127
SMALLINT	2	-32768	32767
MEDIUMINT	3	-8388608	8388607
INT o INTEGER	4	-2147483648	2147483647
BIGINT	8	-9223372036854775808	9223372036854775807

VALORES SIN SIGNO

Ahora bien: existe la posibilidad de duplicar el límite de valor máximo positivo de cada tipo de dato, si eliminamos la posibilidad de almacenar valores negativos.

Pensemos en los ejemplos anteriores: la edad no tiene sentido que sea negativa, entonces, si eliminamos la posibilidad de que ese campo almacene valores negativos, duplicaríamos el limite positivo de almacenamiento, y el campo de tipo TINYINT que normalmente permitía almacenar valores del -128 al 127, ahora dejará almacenar valores desde el 0 hasta el 255.

Esto puede ser útil para almacenar precios, cantidades de objetos o magnitudes que no puedan ser negativas, etc.

Tipo de dato	Bytes	Valor mínimo	Valor máximo
TINYINT	1	0	255
SMALLINT	2	0	65535
MEDIUMINT	3	0	16777215
INT o INTEGER	4	0	4294967295
BIGINT	8	0	18446744073709551615

NÚMEROS CON DECIMALES

Dejemos los enteros y pasemos ahora a analizar los valores numéricos con decimales.

Estos tipos de datos son necesarios para almacenar precios, salarios, importes de cuentas bancarias, etc. que no son enteros.

Tenemos que tener en cuenta que si bien estos tipos de datos se llaman "de coma flotante", por ser la coma el separador entre la parte entera y la parte decimal, en realidad MySQL los almacena usando un punto como separador.

En esta categoría, disponemos de tres tipos de datos: FLOAT, DOUBLE y DECIMAL.

La estructura con la que podemos declarar un campo FLOAT implica definir dos valores: la longitud total (incluyendo los decimales y la coma), y cuántos de estos dígitos son la parte decimal. Por ejemplo:

FLOAT (6.2)

Esta definición permitirá almacenar como mínimo el valor -999.99 y como máximo 999.99 (el signo menos no cuenta, pero el punto decimal sí, por eso son seis dígitos en total, y de ellos dos son los decimales).

La cantidad de decimales (el segundo número entre los paréntesis) debe estar entre 0 y 24, ya que ése es el rango de precisión simple.

En cambio, en el tipo de dato DOUBLE, al ser de doble precisión, sólo permite que la cantidad de decimales se defina entre 25 y 53.

Debido a que los cálculos entre campos en MySQL se realizan con doble precisión (la utilizada por DOUBLE) usar FLOAT, que es de simple precisión, puede traer problemas de redondeo y pérdida de los decimales restantes.

Por último, DECIMAL es ideal para almacenar valores monetarios, donde se requiera menor longitud, pero la "máxima exactitud" (sin redondeos).

Este tipo de dato le asigna un ancho fijo a la cifra que almacenará.

El máximo de dígitos totales para este tipo de dato es de 64, de los cuales 30 es el número de decimales máximo permitido. Más que suficientes para almacenar precios, salarios y monedas.

El formato en el que se definen en el phpMyAdmin es idéntico para los tres: primero la longitud total, luego, una coma y, por último, la cantidad de decimales.

Datos alfanuméricos

Para almacenar datos alfanuméricos (cadenas de caracteres) en MySQL poseemos los siguientes tipos de datos:

CHAR

Comencemos por el tipo de dato alfanumérico más simple: CHAR (character, o caracter).

Este tipo de dato permite almacenar textos breves, de hasta 255 caracteres de longitud como máximo en caracteres que le definamos, aunque no lo utilicemos.

Por ejemplo, si definiéramos un campo "nombre" de 14 caracteres como CHAR, reservará (y consumirá en disco) este espacio.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
J	u	a	n		P	e	r	e	z				
C	a	r	l	o	s		G	a	r	c	i	a	
J	o	s	e		R	a	m	i	r	e	z		
L	u	i	s		F	e	r	n	a	n	d	e	z
P	e	p	e		L	o	p	e	z				

Por lo tanto, no es eficiente cuando la longitud del dato que se almacenará en un campo es desconocida a priori (típicamente, datos ingresados por el usuario en un formulario, como su nombre, domicilio, etc.)

¿En qué casos usarlo, entonces? Cuando el contenido de ese campo será completado por nosotros, programadores, al agregarse un registro y, por lo tanto, estamos seguros de que la longitud siempre será la misma.

Pensemos en un formulario con botones de radio para elegir el "sexo"; independientemente de lo que muestren las etiquetas visibles para el usuario, podríamos almacenar un solo carácter M o F (masculino o femenino) y, en consecuencia, el ancho del campo CHAR podría ser de un digito, y sería suficiente. Lo mismo sucede con códigos que identifiquen provincias, países, estados civiles, etc.

VARCHAR

Complementariamente, el tipo de dato VARCHAR (charactervarying, o caracteres variables) es útil cuando la longitud del dato es desconocida, cuando depende de la información que el usuario escribe en campos o áreas de texto de un formulario.

La longitud máxima permitida era de 255 caracteres hasta MySQL 5.0.3. pero desde esta versión cambio a un máximo de 65.535 caracteres.

Este tipo de dato tiene la particularidad de que cada registro puede tener una longitud diferente, que dependerá de su contenido; si en su registro el campo "nombre" (supongamos que hubiera sido definido con un ancho máximo de 20 caracteres) contiene solamente el texto: "Pepe", consumirá sólo cinco caracteres, cuatro para las cuatro letras, y uno más que indicará cuántas letras se utilizaron.

Si luego, en otro registro, se ingresa un nombre de 15 caracteres, consumirá 16 caracteres (siempre uno más que la longitud del texto, mientras la longitud no supere los 255 caracteres; si no los supera, serán dos los bytes necesarios para indicar la longitud).

Por lo tanto, será más eficiente para almacenar registros cuyos valores tengan longitudes variables, ya que si bien "gasta" uno o dos caracteres por registro para declarar la longitud, esto le permite ahorrar muchos otros caracteres que no serían utilizados.

En cambio, en el caso de datos de longitud siempre constante, sería un desperdicio gastar un carácter por registro para almacenar la longitud, y por eso convendría utilizar CHAR en esos casos.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
J	u	a	n		P	e	r	e	z				
C	a	r	l	o	s		G	a	r	c	i	a	
J	o	s	e		R	a	m	i	r	e	z		
l	u	i	s		F	e	r	n	a	n	d	e	z
P	e	p	e		L	o	p	e	z				

TEXT

Antes de la versión 5.0.3. de MySQL, este campo era el utilizado "por excelencia" para descripciones de productos, comentarios, textos de noticia, y cualquier otro texto largo.

Pero, a parir de la posibilidad de utilizar VARCHAR para longitudes de hasta 65.535 caracteres, es de esperar que se utilice cada vez menos este tipo de campo.

La principal desventaja de TEXT es que no puede indexarse fácilmente (a diferencia de VARCHAR).

Tampoco se le puede asignar un valor predeterminado a un campo TEXT (un valor por omisión que se complete automáticamente si no se ha proporcionado un valor al insertar un registro).

Sólo deberíamos utilizarlo para textos realmente muy largos, como los que mencionamos al comienzo de este párrafo.

BLOB

Es un campo que guarda información en formato binario y se utiliza cuando desde PHP se almacena en la base de datos el contenido de un archivo binario (típicamente, una imagen o

un archivo comprimido ZIP) leyéndolo byte a byte, y se requiere almacenar todo su contenido para luego reconstruir el archivo y servidor al navegador otra vez, sin necesidad de almacenar la imagen o el ZIP en un disco, sino que sus bytes quedan guardados en un campo de una tabla de la base de datos.

El tamaño máximo que almacena es de 65.535 bytes.

De todos modos, y como lo hemos mencionado en este ejemplo, respecto al tipo de dato para una imagen, usualmente no se guarda "la imagen" (sus bytes, el contenido del archivo) en la base de datos porque, un sitio grande, se vuelve muy pesada y lenta la base de datos, sino que almacena sólo la URL que lleva hasta la imagen.

De esa forma, para mostrar la imagen simplemente se lee ese campo URL y se completa una etiqueta img con esa URL, y esto es suficiente para que el navegador muestre la imagen. Entonces, con un VARCHAR alcanza para almacenar la URL de una imagen.

El campo BLOB es para almacenar directamente "la imagen" (o un archivo comprimido, o cualquier otro archivo binario), no su ruta.

TINYBLOB, MEDIUMBLOB Y LONGBLOB

Similares al BLOB, sólo cambia la longitud máxima:

TINYBLOB es de 255 bytes

MEDIUMBLOB es de 16.777.215 bytes, y

LONGBLOB es de 4 Gb (o lo máximo que permita manipular el sistema operativo).

Datos de fecha y hora

En MySQL, poseemos varias opciones para almacenar datos referidos a fechas y horas.

Veamos las diferencias entre uno y otro, y sus usos principales, así podemos elegir el tipo de dato apropiado en cada caso.

DATE

El tipo de dato DATE nos permite almacenar fechas en el formato: AAAA-MM-DD (los cuatro primeros dígitos para el año, los dos siguientes para el mes, y los últimos dos para el día).

Atención:

En los países de habla hispana estamos acostumbrados a ordenar las fechas en Día, Mes y Año, pero paraMySQL es exactamente al revés.

Tengamos en cuenta que esto nos obligará a realizar algunas maniobras de reordenamiento utilizando funciones de manejo de caracteres.

Si bien al leer un campo DATE siempre nos entrega los datos separados por guiones, al momento de insertar un dato nos permite hacerlo tanto en formato de número continuo (por ejemplo, 201512319, como utilizando cualquier carácter separador (2015-12-31 o cualquier otro carácter que separe los tres grupos).

El rango de fechas que permite manejar desde el 1000-01-01 hasta el 9999-12-31.

Es decir, que no nos será útil si trabajamos con una línea de tiempo que se remonta antes del año 1000, (¿alguna aplicación relacionada con la historia?), pero si nos resultara útil para datos de un pasado cercano y un futuro muy largo por delante, ya que llega casi hasta el año 10.000.

DATETIME

Un campo definido como DATETIME nos permitirá almacenar información acerca de un instante de tiempo, pero no sólo la fecha sino también su horario, en el formato:

AAAA-MM-DD HH:MM:SS

Siendo la parte de la fecha de un rango similar al del tipo DATE (desde el 1000-01-01 00:00:00 al 9999-12-31 23:59:59), y la parte del horario, de 00:00:00 a 23:59:59.

TIME

Este tipo de cambio permite almacenar horas, minutos y segundos, en el formato HH:MM:SS, y su rango permitido va desde -839:59:59 hasta 839:59:59 (unos 35 días hacia atrás y hacia adelante de la fecha actual). Esto lo hace ideal para calcular tiempos transcurridos entre dos momentos cercanos.

TIMESTAMP

Un campo que tenga definido el tipo de dato TIMESTAMP sirve para almacenar una fecha y un horario, de manera similar a DATETIME, pero su formato y rango de valores serán diferentes.

El formato de un campo TIMESTAMP puede variar entre tres opciones:

AAAA-MM-DD HH:MM:SS

AAAA-MM-DD

AA-MM-DD

Es decir, la longitud posible puede ser de 14, 8 o 6 dígitos, según qué información proporcionemos.

El rango de fechas que maneja este campo va desde el 1970-01-01 hasta el año 2037.

Además, posee la particularidad de que podemos definir que su valor se mantenga actualizado automáticamente, cada vez que se inserte o que se actualice un registro.

De esa manera, conservaremos siempre en ese campo la fecha y hora de la última actualización de ese dato, que es ideal para llevar el control sin necesidad de programar nada.

YEAR

En caso de definir un campo como YEAR, podremos almacenar un año, tanto utilizando dos como cuatro dígitos.

En caso de hacerlo en dos dígitos, el rango posible se extenderá desde 70 hasta 99 (del 70 hasta el 99 se entenderá que corresponden al rango de años entre 1970 y 1999, y del 00 al 69 se entenderá que se refiere a los años 2000 a 2069); en caso de proporcionar los cuatro dígitos, el rango posible se ampliará, yendo desde 1901 hasta 2155.

Una posibilidad extra, ajena a MySQL pero relativa a las fechas y horarios, es generar un valor de timestamp con la función time de PHP (repito, no estamos hablando de MySQL, no nos confundamos a causa de tantos nombres similares).

A ese valor, lo podemos almacenar en un campo INT de 10 dígitos.

De esa forma, será muy simple ordenar los valores de ese campo (supongamos que es la fecha de una noticia) y luego podremos mostrar la fecha transformando ese valor de timestamp en algo legible mediante funciones de manejo de fecha propias de PHP.

Atributos de los campos

Ya hemos visto los diferentes tipos de datos que es posible utilizar al definir un campo en una tabla, pero estos tipos de datos pueden poseer ciertos modificadores o "atributos" que se pueden especificar al crear el campo, y que nos brindan la posibilidad de controlar con mayor exactitud qué se podrá almacenar en ese campo, cómo lo almacenaremos y otros detalles.

Aunque algunos de estos atributos ya los hemos utilizado intuitivamente al pasar en algunos de los ejemplos anteriores, a continuación, vamos a analizar más en detalle.

¿NULL O NOT NULL?

Algunas veces tendremos la necesidad de tener que agregar registros sin que los valores de todos sus campos sean completados, es decir, dejando algunos campos vacíos (al menos provisoriamente).

Por ejemplo, en un sistema de comercio electrónico, podría ser que el precio, o la descripción completa de un producto, o la cantidad de unidades en depósito, o la imagen del producto, no estén disponibles en el momento en que, como programadores, comencemos a trabajar con la base de datos.

Todos esos campos, nos conviene que sean definidos como NULL (nulos), para que podamos ir agregando registros con los datos básicos de los productos (su nombre, código, etc.) aunque todavía la gente del área comercial no haya definido el precio, ni el área de marketing haya terminado las descripciones, ni los diseñadores hayan subido las fotos (es típica esta división de tareas en empresas grandes, y hay que tenerla presente, porque afecta la declaración de campos de nuestras tablas).

Si definimos esos campos que no son imprescindibles de llenar de entrada como NULL (simplemente marcando la casilla de verificación a la altura de la columna NULL, en el phpMyAdmin), el campo queda preparado para que, si no es que proporcionado un valor, quede vacío pero igual nos permita completar la inserción de un registro completo.

Por omisión, si no marcamos ninguna casilla, todos los campos son NOT NULL, es decir, es obligatorio ingresar algún valor en cada campo para poder cargar un nuevo registro en la tabla.

VALOR PREDETERMINADO (DEFAULT)

Muchas veces necesitamos agilizar la carga de datos mediante un valor por defecto (default).

Por ejemplo, pensemos en un sistema de pedidos, donde, al llegar el pedido a la base de datos, su estado sea "recibido", sin necesidad de que el sistema envíe ningún valor, sólo por agregar el registro, ese registro debería contener en el campo "estado" el valor de "recibido".

Este es un típico caso de valor predeterminado o por default.

PRIMARY KEY Y AUTO_INCREMENT

Siempre, en toda la tabla, uno de los campos (por convención, el primero, y también por convención usualmente llamado id –por "identificador"-), debe ser de definido como clave primario o Primary Key.

Esto impedirá que se le inserten valores repetidos y que se deje nulo su valor.

Habitualmente, se especifica que el campo elegido para clave primaria sea numérico, de tipo entero (en cualquiera de sus variantes, según la cantidad de elementos que se identificarán) y se le asigna otro atributo típico, que es Auto_Increment, es decir, que no nos preocupamos por darle valor a ese campo: al agregar un registro, MySQL se ocupa de incrementar en uno el valor de la clave primaria del último registro agregado, y se lo asigna al nuevo registro.

Este campo no suele tener ninguna relación con el contenido de la tabla, su objetivo es simplemente identificar cada registro de forma única, irrepetible.

Clave primaria con auto_increment campo en MySQL

Podemos definir un sólo campo como clave primaria, o dos o más campos combinados.

En caso de haber definido dos o más campos para que juntos formen el valor único de una clave primaria, diremos que se trata de una clave primaria "combinada" o "compuesta".

UNIQUE

Si especificamos que el valor de un campo sea Unique, estaremos obligando a que su valor no pueda repetirse en más de un registro, pero no por eso el campo se considerará clave primaria de cada registro.

Esto es útil para un campo que guarde, por ejemplo, número de documentos de identidad, la casilla de correo electrónico usada para identificar el acceso de un usuario, un nombre de usuario, o cualquier otro dato que no debamos permitir que se repita.

Los intentos por agregar un nuevo registro que contenga un valor ya existente en ese campo, serán rechazados.

Componentes de SQL

El lenguaje de consulta estructurado(SQL) está compuesto por comandos, clausulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

Comandos

Los principales tipos de comandos SQL son los siguientes:

Los DDL (Data Definition Languaje) que permiten crear y definir nuevas bases de datos, campos e índices.

Los DML (Data Manipulation Languaje) que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Lenguaje de definición de datos (DDL)

El lenguaje de definición de datos (en inglés Data Definition Languaje), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Incluye órdenes para modificar, borrar o definir las tablas en las que se almacenan los datos. Existen cuatro operaciones básicas: CREATE, ALTER, DROP Y TRUNCATE.

DDL	
Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e indicies
DROP	Empleado para eliminar tablas e índices
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos

CREATE

Este comando crea un objeto dentro del gestor de base de datos, puede ser una base de datos, una tabla.

Ejemplos:

```
#Con el siguiente comando crea una base de datos.

CREATE DATABASE cac2021;

#Con el siguiente comando se crea una tabla en la base de datos.

CREATE TABLE alumnos2021

(

Nombre VARCHAR(25),
```

Apellido VARCHAR(25),

Edad INT,

Curso VARCHAR(25)

)

DROP

Este comando elimina un objeto de la base de datos. Puede ser una tabla, una base de datos.

Ejemplo:

#Con el siguiente comando se elimina la tabla alumnos2021

DROP TABLE alumnos2021;

ALTER

Este comando permite modificar la estructura de un objeto. Se pueden agregar/quitar campos de una tabla, modificar el tipo de un campo.

#Con el siguiente comando se agrega una columna a la base de datos "comision" del tipo INT.

ALTER TABLE alumnos2021 **ADD** comisión **INT**;

#Con el siguiente comando se elimina un campo

ALTER TABLE alumnos2021 **DROP** comision;

Lenguaje de manipulación de datos DML (Data Manipulation Languaje)

Un lenguaje de manipulación de datos (Data Manipulation Languaje) proporcionado por el sistema de gestión de base de datos permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado

DML	
Comando	Descripción
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados utilizado para modificar las tablas agregando campos o cambiando la definición de los campos
DELETE	Utilizado para eliminar registros de una tabla

INSERT

Una sentencia INSERT de SQL agrega uno o más registros a una (y solo una) tabla en una base de datos relacional.

Ejemplo forma básica:

INSERT INTO alumnos2021 (Nombre, Apellido, Edad, Curso) **VALUES** ('marcos', 'sanchez', '10', '2169');

Las cantidades de columnas y valores deben ser iguales. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados por la sentencia INSERT deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agregará la fila y devuelve ERROR.

Ejemplo forma sin especificar campos:

INSERT INTO alumnos2021 **VALUES** (‘alejandro’,’gutierrez’,’21’,’2162);

UPDATE

Una sentencia UPDATE de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

Ejemplos:

#el siguiente ejemplo, actualiza todos los campos de la tabla comision

UPDATE alumnos2021 **SET** Curso = ‘2164’;

DELETE

Una sentencia DELETE de SQL es utilizada para eliminar uno o más registros existentes en una tabla de una base de datos.

Ejemplos:

#el siguiente ejemplo, elimina todos los campos de la tabla comision

DELETE FROM alumnos2021;

Tanto para para los comandos UPDATE y DELETE, se necesitan clausulas específicas para que no se actualicen o eliminen todos los registros.

Clausulas

Las clausulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular

CLAUSULAS	
Comando	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros.
WHERE	Utilizada para determinar los registros seleccionados en la

	cláusula FROM
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos.

Operadores

Operadores lógicos

Operadores lógicos	
Operador	uso
AND	Es el "y" lógico. Evalua dos condiciones y devuelve un valor de verdad solo si ambas son ciertas
OR	Es el "o" lógico. Evalua dos condiciones y devuelve un valor de verdad si alguna de las dos es ciertas
NOT	Negación lógica. Devuelve el valor contrario de la expresion

Operadores comparación.

Operadores comparación	
Operador	uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor o igual que
>=	Mayor o igual que
BETWEEN	Intervalo
LIKE	Comparación
IN	Especificar

Funciones

Las funciones se usan dentro de una clausula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros

Comando	descripción

AVG	Utilizada para calcular el promedio de los valores de un campo determinado
COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un campo especifico
MIN	Utilizada para devolver el valor más bajo de un campo especifico

Consultas: sentencia SELECT

La sentencia **SELECT** "selecciona" los campos que conformaran la consulta, es decir, que establece los campos que se visualizaran o compondrán la consulta.

El parámetro "listas_campo" este compuesto por uno o más nombres de campos, separados por comas, pudiéndose especificar también el nombre de la tabla a la cual pertenecen, seguido de un punto y del nombre del campo correspondiente.

Si se desea seleccionar todos los campos de una tabla, se puede utilizar el asterisco (*) para indicarlo.

Ejemplos:

SELECT * **FROM** alumnos2021;

SELECT Nombre, Apellido **FROM** alumnos2021;

Una sentencia **SELECT** no puede escribirse sin las clausula **FROM**.

Una clausula es una extensión de un mandato que complementa a una sentencia o instrucción, pudiendo complementar también a otras sentencias.

En este caso, la cláusula FROM permite indicar en que tablas o en qué consultas (queries) se encuentran los campos especificados en las sentencias **SELECT**.

Clausula WHERE

La cláusula WHERE es opcional, y permite seleccionar qué registros aparecerán en la consulta (si no se especifica aparecerán todos los registros).

Para indicar este conjunto de registros se hace uso de criterios o condiciones, que no es más que una comparación del contenido de un campo con un determinado valor (este valor puede ser constante (valor predeterminado), el contenido de un campo, una variable, un control, etc.).

Ejemplo:

SELECT * **FROM** alumnos2021 **WHERE** Nombre = 'marcos';

El ejemplo anterior selecciona todos los campos de la tabla alumnos2021, pero los registros de todos los alumnos que se llamen ‘marcos’.

Ejemplo:

SELECT * **FROM** alumnos2021 **WHERE** Nombre = ‘marcos’ OR Nombre = ‘juan’;

El ejemplo anterior selecciona todos los campos de la tabla alumnos2021, pero los registros de todos los alumnos que se llamen ‘marcos’ o ‘juan’;

Ejemplo:

SELECT * **FROM** alumnos2021 **WHERE**Edad >=18;

El ejemplo anterior selecciona todos los campos de la tabla alumnos2021, pero los registros que la edad sea mayor o igual a 18.

Ejemplo:

SELECT * **FROM**alumnos2021 **WHERE** Edad>=18 **AND** Edad<=45
Selecciona todos los alumnos con edades comprendidas entre los 18 y los 45 años.

Ordenar los registros.

Se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula ORDER BY.

Ejemplo:

SELECT nombre **FROM** alumnos2021 **ORDER BY** nombre;

También se pueden ordenar los registros por más de un campo.

SELECT Nombre, edad **FROM** alumnos2021 **ORDER BY** nombre;

Y se puede especificar el orden de los registros, ascendente mediante la cláusula (ASC-se toma este valor por defecto) o descendente (DESC)

SELECT nombre, edad **FROM** alumnos2021 **ORDER BY** nombre **DESC**;

Operador LIKE

Se utiliza para comparar

SELECT * **FROM**alumno2021 **WHERE** Nombre **LIKE** 'al%'
Selecciona los alumnos cuyo nombre comience con los caracteres 'al'.

SELECT * **FROM** alumnos2021 **WHERE** apellidos **LIKE** ‘%ez';
Selecciona los alumnos cuyos apellidos terminen con los caracteres 'ez'.

SELECT * **FROM** alumnos2021 **WHERE** apellidos **LIKE** ‘%zamo%'
Selecciona los alumnos cuyos apellidos contengan, en cualquier posición, los caracteres 'zamo'.

Eliminar Registros específicos

Para eliminar los registros de una tabla usamos el comando DELETE

DELETE FROM alumnos2021;

La ejecución del comando indicado en la línea anterior borra TODOS los registros de la tabla.

Si queremos eliminar uno o varios registros debemos indicar cuál o cuáles, para ello utilizamos el comando DELETE junto con la cláusula WHERE con la cual establecemos la condición que deben cumplir los registros a borrar. Por ejemplo, queremos eliminar aquel registro cuyo nombre de alumno es 'Leonardo':

DELETEFROM alumnos2021 **WHERE** nombre='marcos';

Si solicitamos el borrado de un registro que no existe, es decir, ningún registro cumple con la condición especificada, no se borrarán registros.

Actualizar registros específicos

Para modificar uno o varios datos de uno o varios registros utilizamos UPDATE (actualizar).

Por ejemplo, en nuestra tabla alumnos2021, queremos cambiar los valores de todas las comisiones, por "2180":

UPDATE usuarios **SET** comision='2180';

Utilizamos **UPDATE** junto al nombre de la tabla y **SET** junto con el campo a modificar y su nuevo valor.

El cambio afectará a todos los registros.

Podemos modificar algunos registros, para ello debemos establecer condiciones de selección con WHERE.

Por ejemplo, queremos cambiar el valor correspondiente a la clave de nuestro usuario llamado

'marcos', queremos como nueva comisión '2164', necesitamos una condición WHERE que afecte solamente a este registro:

UPDATE alumnos2021 set comisión='2164' **WHERE** nombre='marcos';

Si no encuentra registros que cumplan con la condición del WHERE, ningún registro es afectado.

Las condiciones no son obligatorias, pero si omitimos la cláusula **"where",** la actualización afectará a todos los registros.

También se puede actualizar varios campos en una sola instrucción:

UPDATE alumnos2021 **SET** nombre='miguel', comisión='2168' **WHERE** nombre='marcos';

Para ello colocamos UPDATE, el nombre de la tabla, SET junto al nombre del campo y el nuevo valor yseparado por coma, el otro nombre del campo con su nuevo valor.

Trabajar con varias tablas (joins)

Constraints o restricciones

Para asegurar la integridad de los datos almacenados en nuestras tablas, podemos crear restricciones, algunos los hemos utilizado sin querer o simplemente desconocemos que lo que hicimos fue una restricción. Estas restricciones las podemos implementar al momento de crear las tablas o modificarlas.

Los principales tipos de restricciones que existen son:

-PRIMARY KEY

-UNIQUE

-FOREING KEY

PRIMARY KEY

Es la más común de todas debido a que cada una de nuestras tablas debe ser completamente relacional y para lograr esto siempre debe existir una llave primaria dentro de cada tabla que identifique cada fila como única.

Existen ciertos requerimientos para la creación de una llave primaria:

-La columna utilizada en una restricción PRIMARY KEY, no puede aceptar NULL

-No pueden repetir los valores en la columna, debe ser único

-Solamente puede existir una restricción de tipo PRIMARY KEY por cada tabla

UNIQUE

Este tipo de restricción es muy parecida a PRIMARY KEY, las diferencias son las siguientes:

-También genera un índice automáticamente.

-La tabla puede tener más de una restricción UNIQUE

-Si puede aceptar NULL, pero solo una fila puede contenerlo ya que como su nombre lo indica, es de tipo UNIQUE o único.

FOREIGN KEY

Se forma de una columna o la combinación de varias columnas de una tabla que sirve como enlace hacia otra tabla donde en esta última, dicho enlace son la o las columnas que forman la PRIMARY KEY. Las columnas involucradas como llave foránea deben tener el mismo tipo de datos que la llave primaria de la tabla relacionada. Una llave foránea no cra un índice automáticamente, por lo que se recomienda generar uno para incrementar el rendimiento de la consulta

Requerimientos para la restricción FOREIGN KEY:

-Los valores ingresados en la columna de la llave foránea, debe existir en la tabla a la que se hace referencia en la columna de la llave primaria

-Solo se pueden hacer referencia a las llaves primarias de tablas que se encuentren dentro de la misma base de datos

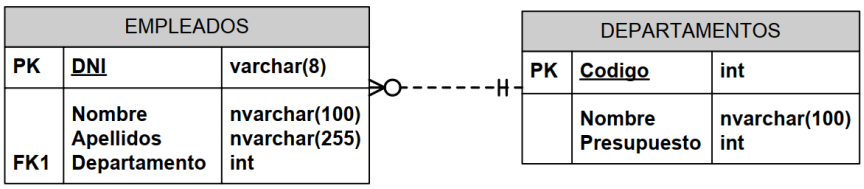
-Solo puede hacer referencia a columnas de restricciones PRIMARY KEY o UNIQUE

Para entrar en tema e ir aplicando los nuevos conceptos tengamos como referencia la siguiente tabla de empleados ya que hasta ahora hemos trabajado con una sola tabla, pero en general, se trabaja con varias tablas.

EMPLEADOS	
DNI	varchar(8)
Nombre	varchar(25)
Apellidos	varchar(25)
Codigo_depa	INT
Nombre_depa	varchar(25)
Prespuesto	INT

Por ejemplo, los datos de nuestra tabla “empleados” podrían separarse en 2 tablas, unos empleados y otra "departamentos", de este modo, evitamos almacenar tantas veces los nombres de los departamentos en la tabla "empleados”, así como también cometer errores de escritura al momento de ingresar el dato concreto

Ahora con esta nueva división deberemos pensar en dos conceptos para los campos, clave primaria y clave foránea, ya que con identificar las mismas podremos crear las relaciones de las tablas y para ello debemos utilizar los JOINS.

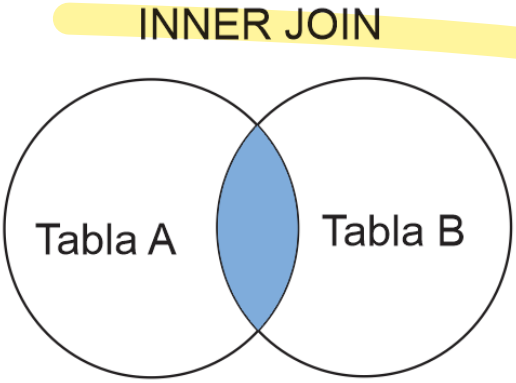


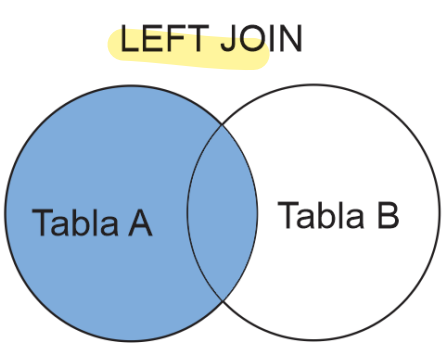
Joins

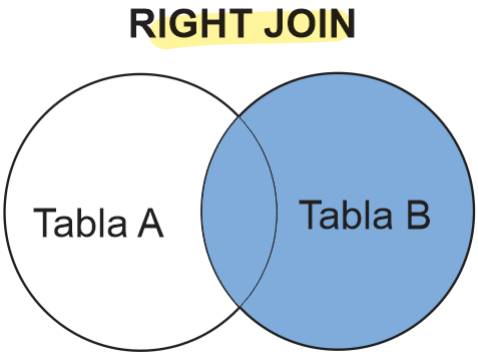
Joins significa uniones y tal como su significado nos indica, nos permite hacer uniones entre tablas y los resultados de las consultas a las mismas lo cual nos va a permitir proveer ciertas flexibilidades adicionales además su sintaxis es mucho más utilizada, también ganamos una mayor performance.

INNER JOIN: En esta sentencia solo se muestran los registros que coinciden tanto en la tabla A como la tabla B

LEFT JOING – RIGHT JOIN: Estos tipos de JOINS no excluyen resultados de alguna de las dos tablas, si hubiese campos que no coinciden podríamos incluirlos en el resultado mediante LEFT o RIGHT JOIN.







Ejemplo:

SELECT nombre, apellido **FROM** empleados **INNER JOIN** departamentos

Si bien ya pudimos cruzar ambas tablas, aún nos falta aclarar dónde esta ese cruce, para ello deberemos hacerlo mediante la clave primaria (PK) de una tabla con la clave foránea (FK) de la tabla relacionada

La sintaxis del join no utiliza el WHERE, sino que requiere la palabra ON. Es ahí en donde indicaremos el filtro a tener en cuenta para realizar el cruce.

Ejemplo

SELECT nombre, apellido **FROM** empleados **INNER JOIN** departamentos **ON** empleados.departamento = departamentos.codigo