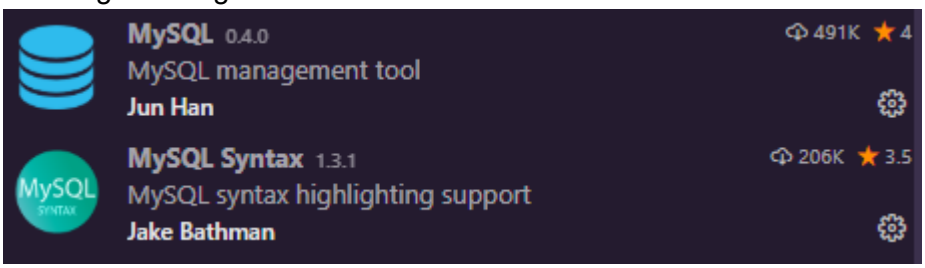
Instalación MySQL Server e integración con Visual Studio Code.

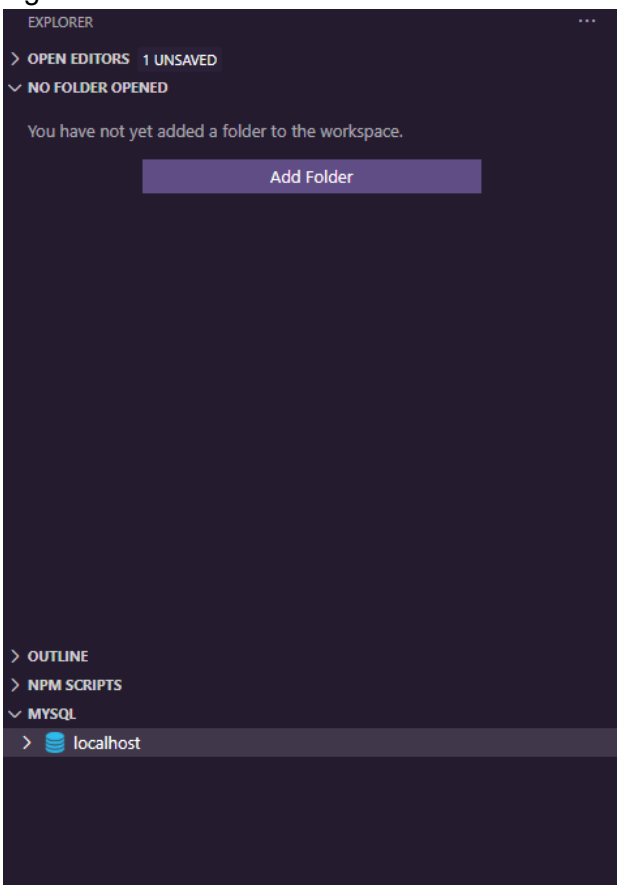
1) Descargar el instalador de https://dev.mysql.com/downloads/installer/
2) Ejecutar el instalador y seleccionar Server Only -> Execute.
3) Darle a next hasta llegar a la pantalla Authentication Method: Seleccionar Use Legacy Authentication Method.
4) En la siguiente pantalla setear contraseña para el usuario root en MySQL Root Password.
5) Darle next y al llegar a Apply Configuration apretar Execute.

En Visual Studio Code:

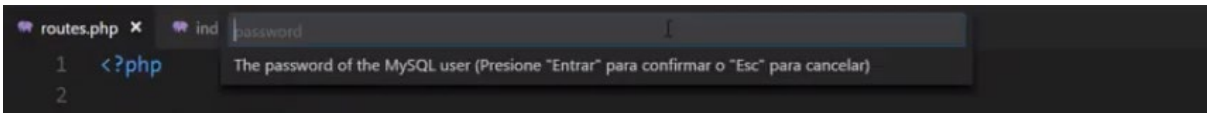
1) Descargar las siguientes extensiones:



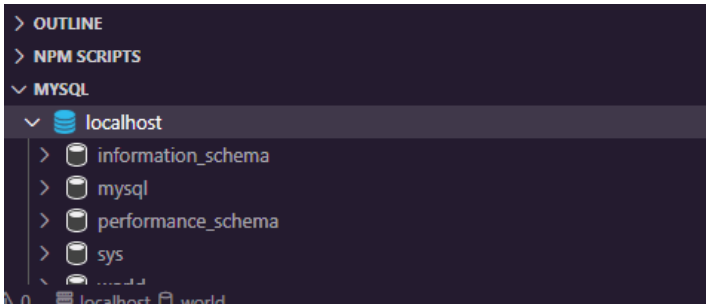
2) Cerrar y volver a abrir Visual Studio Code.
3) Apretar el símbolo + en el apartado MySQL. Al ser la primera vez que se configura no aparecerá ninguna base de datos:



4) Rellenar usuario y contraseña, a los demas datos (puerto y SSL) apretar enter sin modificar nada.



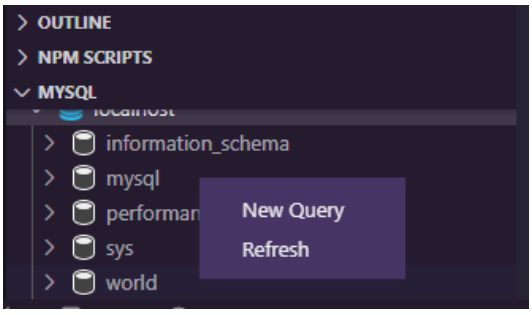
5) Deberá aparecer localhost



6) Descargar world.sql de Google Drive y abrir con Visual Studio Code.

7) Apretar botón derecho -> Run SQL Query.

8) Hacer un refresh (Botón derecho sobre localhost->Refresh). Deberá aparecer la base de datos world dentro de localhost. Para hacer una nueva query apretar botón derecho sobre la base de datos world -> New Query.



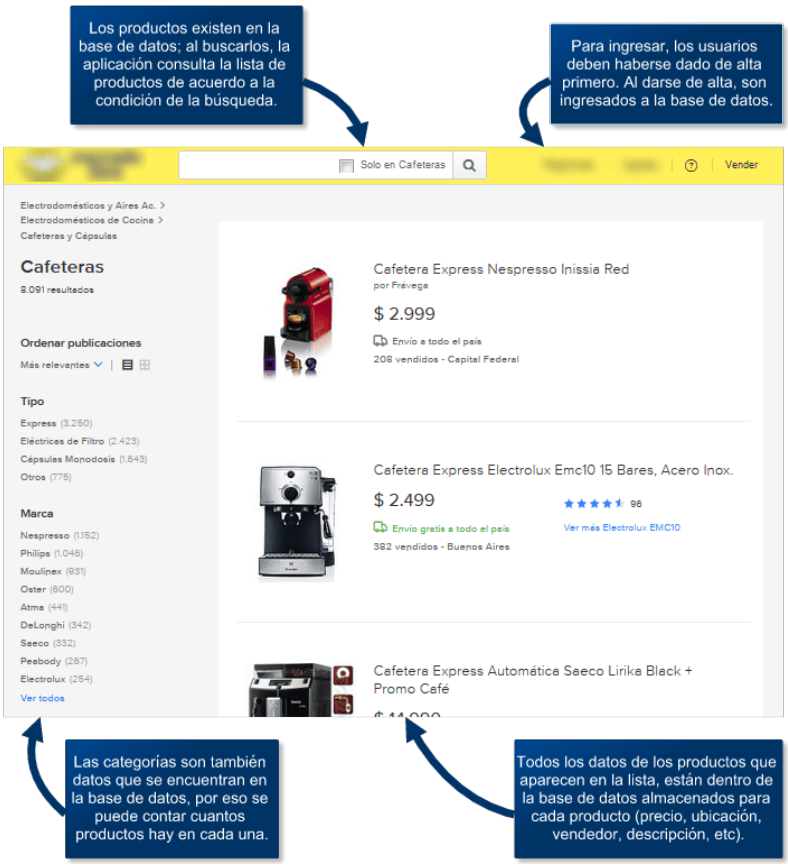
Introducción

Antes de que se conociera el concepto de bases de datos, las aplicaciones (o programas) utilizaban “procesamiento de archivos” para trabajar con los datos que estaban guardados en ellos. Entonces cada usuario definía e implementaba los archivos necesarios para utilizar los datos como parte de la programación de dicha aplicación.

El programador de la aplicación definía los datos que se iban a necesitar para el archivo “Clientes” y creaba lo que se denominaba un “record” o “registro” (que contenía por ejemplo el nombre, la dirección, el CP, el teléfono del cliente). Quizás los datos del producto que había comprado cada cliente los guardaba en otro archivo, el de “Ventas”. Cuando buscaba, por ejemplo, la historia de las ventas de un cierto cliente, debía recorrer ambos archivos, el de clientes para encontrar los datos del cliente y el de ventas para buscar sus ventas.
Así, se recorría en forma secuencial (desde el comienzo, registro por registro) cada archivo y se iba mostrando la información a medida que se la encontraba. Si había que realizar cálculos, debían usarse variables incrementales para ir sumando las ventas de cada cliente.
Un proceso bastante largo y que generalmente llevaba mucho tiempo.
Veamos otro ejemplo....

I - Estructura de una base de datos

Veamos ahora otro ejemplo en donde se manejan datos



II - Estructura de una base de datos

Utilizando el procesamiento de archivos, antes de poder trabajar en la aplicación del reporte, el programador debe saber dónde se encuentra el archivo de la aplicación de ventas, en qué máquina y en qué disco y carpeta de dicha máquina se encuentra guardado. Él desarrolla su aplicación y todo anda bien hasta que un día el desarrollador del programa de ventas decide introducir una mejora. Como parte de la misma, decide mover el archivo a otra carpeta dentro del disco.

Así, las bases de datos se crearon para poder tener una independencia entre las aplicaciones o programas y los archivos donde están almacenados los datos, es decir, su ubicación física en los discos rígidos.

Otro motivo importantísimo que afectaba la operatoria en la época del procesamiento de archivos era la imposibilidad de poder acceder al archivo al mismo tiempo para operar, en nuestro ejemplo, ambas aplicaciones. Es decir, si mientras se estaba obteniendo el reporte, leyendo el archivo de ventas, un usuario del sistema de ventas trataba de ingresar una venta, dicho sistema arrojaba un error de acceso indicando que el archivo estaba siendo abierto por otro usuario. Es decir que el procesamiento era monousuario, no multiusuario.

DBMS o motor de la base de datos

Un DBMS o motor de una base de datos es una colección de programas que permite a los usuarios crear y mantener una base de datos. Es un sistema de software de propósito general que facilita la definición, construcción, manipulación y compartición de bases de datos entre usuarios y aplicaciones.

El DBMS deberá tener algún “mecanismo de control de concurrencia”, ya que debe permitir que muchos usuarios accedan al mismo tiempo a trabajar con los datos. O sea algunos podrían estar generando un reporte mientras otros están agregando nuevos datos (por ejemplo ingresando una venta).

El DBMS debe permitir que ambos usuarios trabajen y que los datos sean consistentes para ambas acciones.
En una base de datos existen diferentes tipos de usuarios de acuerdo a las actividades que pueden realizar en ella:

Administrador (DBA)

Es el usuario con más poderes dentro de la base de datos. Se ocupa de:

- administrar los permisos de acceso a los demás usuarios,
- que el DBMS funcione correctamente,
- analizar el rendimiento (o velocidad) del motor para que los usuarios puedan trabajar en los niveles esperados,
- administrar los recursos necesarios para asegurarse que tanto las bases de datos como el DBMS se comporten de acuerdo a los niveles previamente determinados.

Diseñador

Es el usuario que trabajó desde el inicio, mucho antes incluso de la existencia de la base de datos misma. El desarrolló los modelos de datos de acuerdo al requerimiento del cliente (el dueño del mini-mundo que se necesitaba modelar) y también determinó cuáles son las diferentes vistas a las que necesitarán acceder los usuarios.

Usuarios finales

Son los usuarios que van a interactuar con las diferentes vistas de la base de datos y para los cuales se la construyó.

Analistas y programadores de aplicaciones

Los analistas son los que realizaron el relevamiento de las necesidades que debía cubrir la base de datos y los programadores de aplicaciones crearon los programas que garantizaron que dichas necesidades pudiesen cumplirse.

Lenguajes DBMS

Una vez que se ha realizado el diseño de la base de datos, luego de elegir el DBMS apropiado, se debe implementar la misma, usando los lenguajes propios del DBMS. Para realizar el diseño, se utiliza el “lenguaje de definición de datos” o DDL por sus siglas en inglés. Mediante este lenguaje se crea la base de datos, todas las estructuras y sus objetos.

En este caso el DBMS utiliza un compilador **DDL** que interpreta dicho lenguaje y procesa las sentencias.

Por último, se utiliza el **DML** o “lenguaje de manipulación de datos” para especificar las vistas de los usuarios finales y los diferentes mapeos a los niveles conceptual y exterior. En los DBMS actuales, el lenguaje utilizado es el mismo, y son las sentencias las que realizan las diferentes tareas. Estas sentencias se clasifican en sentencias **DML** (crear, modificar o borrar datos) y **DDL** (crear, modificar o borrar bases de datos)**.**

Conceptos del Modelo Relacional

Como vimos en la primera parte de esta Unidad, el diseñador arma un esquema conceptual que refleja el sistema o negocio de interés, lo que hemos llamado “el mini-mundo” a modelar. Es decir. antes que nada, debe tener una idea de cuál es ese pequeño entorno que va a modelar, o sea cuál es el alcance del mismo, dónde comienza y dónde termina.

Para saber esto, generalmente los diseñadores se reúnen con el cliente y éste les “cuenta”, con sus palabras, lo que necesita. Es un trabajo arduo que implica mucha capacidad de interpretación por parte de los diseñadores ya que, a menudo, el cliente asume muchas funcionalidades por su amplio conocimiento del negocio y está en la habilidad del diseñador el hacer las preguntas correctas para terminar de entender lo que se desea modelar.

El cliente suele hablar de los datos que necesita para realizar su operación, atándolos a ciertas acciones que tienen que ver con el negocio. Por ejemplo, podría decir: “Tengo una

serie de clientes a quienes vendo mis productos. Mis vendedores los visitan personalmente en sus locales, realizan la venta y luego tienen que mandarles un mail con la lista de los productos que compraron, la cantidad y el monto total de la factura, para que puedan verificar los valores y realizar el pago".

Para "atar" el relato o "relevamiento del negocio" al modelo relacional, se deben encontrar aquellos sustantivos importantes para el cliente dentro del texto. Notarás, además, la existencia de otros sustantivos que en general están asociados a los sustantivos "importantes".

Por ejemplo, cliente y vendedor son sustantivos importantes. Ahora, mail también es un sustantivo pero en el relato dice claramente que hay que enviarle un mail al cliente, es decir nos va a interesar conocer cuál es la dirección de correo electrónico del cliente.

Modelo Entidad Relación

- Los sustantivos importantes son las ENTIDADES del modelo. Los sustantivos menos importantes que corresponden a las entidades (en nuestro ejemplo habíamos visto que el mail era uno de ellos) son los ATRIBUTOS.
- Las entidades interactúan entre sí por medio de acciones, por ejemplo, los VENDEDORES “visitan” LOCALES que “pertenecen a” los CLIENTES para “venderles” los PRODUCTOS. Las acciones son las “relaciones” del modelo.

Las entidades luego contendrán los datos relevantes que hemos indicado son sus atributos. Así en nuestro ejemplo, sabemos que del cliente necesitaremos saber su mail, seguramente también su nombre y dirección o número de teléfono.
Tenemos así los diferentes clientes:

Cliente	Nombre	Dirección	Email	Teléfono
1	J. Pérez	Aráoz 1340	jperez@...com.ar	3302982
2	A. López	Frías 239	alopez@...com.ar	4472992
3	M. Arias	Palpa 2237	marias@...com.ar	4883937

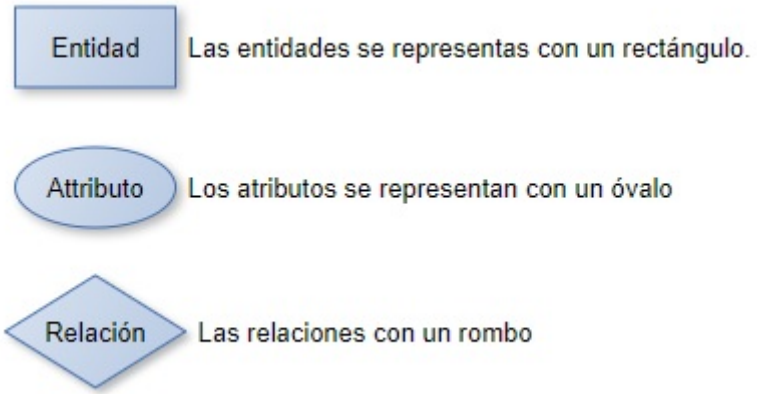
Ahora que sabemos esto, estamos en condiciones de definir qué es el modelo relacional....

Modelo Relacional

El modelo relacional representa a la base de datos como una colección de “entidades” relacionadas entre sí, donde cada entidad se puede ver como una “tabla”.
Podrías pensar en una tabla como si fuera una hoja de cálculo, con sus filas y columnas. Las filas simbolizan cada una de las instancias de la tabla, por ejemplo un cliente, un vendedor o una venta. Las columnas sirven para interpretar el significado de los valores de la fila.

- En el modelo relacional la fila se denomina “T-upla”.
- Las columnas de la tabla son los “atributos”.
    - La tabla, como dijimos anteriormente, se denomina “entidad”.

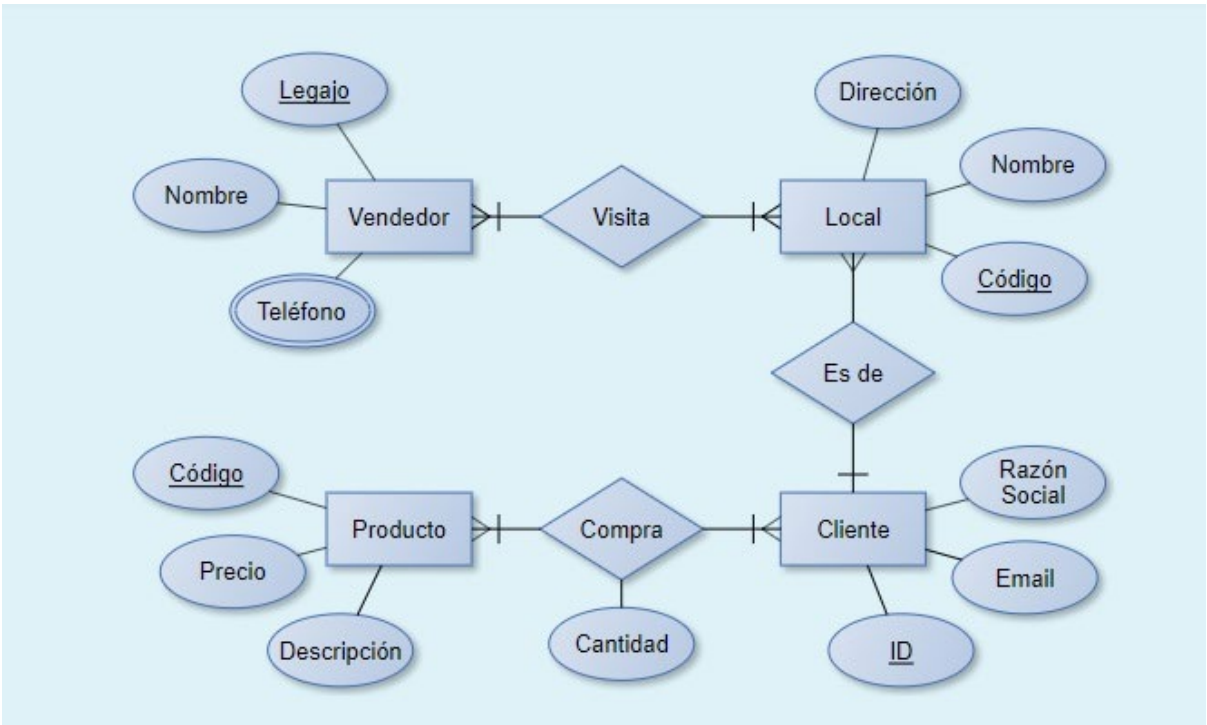
Además como tenemos que construir un diagrama vamos a tener que definir cómo representar cada una de sus partes. Entonces podemos usar la siguiente convención:



Las relaciones y las entidades se conectan por medio de líneas. En el extremo de dichas líneas se encuentra la “cardinalidad” que indica con cuántas instancias (o filas) de la entidad se efectúa la relación

Los diferentes valores que pueden aparecer en una columna o atributo llevan el nombre de “dominio” de ese atributo, y está muy relacionado con el tipo de dato del atributo, es decir si es un número o un texto.

Veamos un ejemplo de cómo sería el modelo del relato de los vendedores...



Notarás que entre clientes y productos hay una relación “compran” que simboliza el hecho de que los clientes van a solicitar ciertos productos a los vendedores. Se puede ver que hay

un atributo en la relación. Es decir el atributo no pertenece a las entidades sino que es propio de la relación.

Bases de datos no relacionales

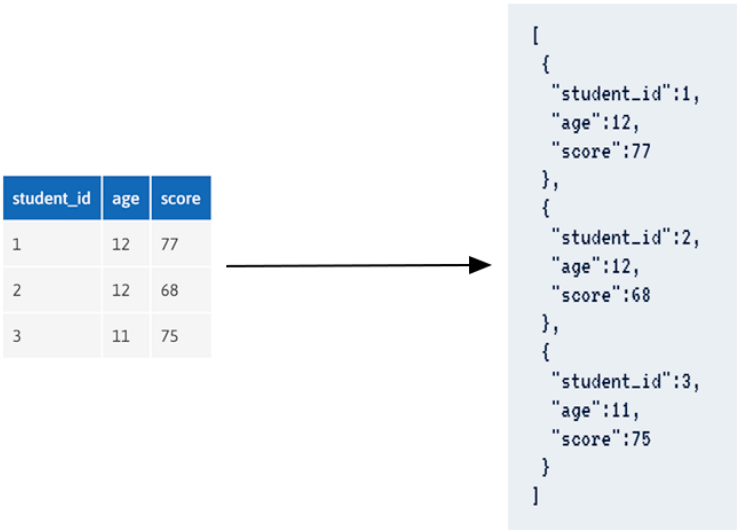
Fuente:

https://aukera.es
https://desarrolloweb.com
https://www.tutorialesprogramacionya.com

Como su propio nombre indica, las bases de datos no relacionales son las que, a diferencia de las relacionales, no tienen un identificador que sirva de relación entre un conjunto de datos y otros. Como veremos, la información se organiza normalmente mediante documentos y es muy útil cuando no tenemos un esquema exacto de lo que se va a almacenar.
La indiscutible reina del reciente éxito de las bases de datos no relacionales es MongoDB seguida por Redis, Elasticsearch y Cassandra.

Formatos

La información puede organizarse en tablas o en documentos. Cuando organizamos información en un Excel, lo hacemos en formato tabla y, cuando los médicos hacen fichas a sus pacientes, están guardando la información en documentos. Lo habitual es que las bases de datos basadas en tablas sean bases de datos relacionales y las basadas en documentos sean no relacionales, pero esto no tiene que ser siempre así. En realidad, una tabla puede transformarse en documentos, cada uno formado por cada fila de la tabla. Solo es una cuestión de visualización.



Los dos esquemas de la imagen contienen exactamente la misma información. Lo único que cambia aquí es el formato: cada documento de la figura de la derecha es una fila de la figura de la izquierda.
Se ve más claro en la tabla, ¿verdad? Lo que pasa es que a menudo en una base de datos no relacional una unidad de datos puede llegar a ser demasiado compleja como para plasmarlo en una tabla. Por ejemplo, en el documento JSON de la imagen que se muestra a

continuación, al tener elementos jerárquicos, es más difícil plasmarlo en una tabla plana. Una solución sería plasmarlo en varias tablas y, por tanto, necesitar de relaciones.

```
[
 {
  "student_id":1,
  "age":12,
  "subjects":{
   "mathematics":{
    "scores":[7,8,7,10],
    "final_score":8
   },
   "biology":{
    "scores":[6,6,5,7],
    "final_score":6
   }
  }
 }
]
```

Esto explica por qué las bases de datos relacionales suelen servirse de tablas y las no relacionales de documentos JSON. En cualquier caso, a día de hoy, las bases de datos más competitivas suelen permitir, de una forma u otra, operaciones de los dos tipos. Por ejemplo, el servicio de base de datos en la nube BigQuery que ofrece Google es, en principio, una base de datos de lenguaje de consulta SQL, por lo que permite fácilmente relacionar varias tablas, pero, a su vez, permite insertar elementos jerárquicos JSON, más propios de bases de datos no relacionales.

La diferencia entre el éxito y el fracaso recae, sobre todo, en el diseño del modelo. Es decir, si se decide que el mejor enfoque es usar una base de datos relacional, no es suficiente con meter la información a lo bruto en una base de datos relacional y esperar a que se relacione sola, porque eso no va a ocurrir. De nada sirve elegir la base de datos más apropiada para nuestro sistema, si luego no se hace un buen diseño.

Creación de bases de datos y tablas.

Una base de datos almacena sus datos en tablas. La misma es creada de la siguiente manera:

CREATE DATABASE databasename;

Una tabla es una estructura de datos que organiza los datos en columnas y filas; cada columna es un campo (o atributo) y cada fila, un registro. La intersección de una columna con una fila, contiene un dato específico, un solo valor.
Cada registro contiene un dato por cada columna de la tabla.
Cada campo (columna) debe tener un nombre. El nombre del campo hace referencia a la información que almacenará.
Cada campo (columna) también debe definir el tipo de dato que almacenará.

nombre	clave
MarioPerez	Marito
MariaGarcia	Mary
DiegoRodriguez	z8080

Gráficamente acá tenemos la tabla usuarios, que contiene dos campos llamados:nombre y clave. Luego tenemos tres registros almacenados en esta tabla, el primero almacena en el campo nombre el valor "MarioPerez" y en el campo clave "Marito", y así sucesivamente con los otros dos registros.

Las tablas forman parte de una base de datos.
Nosotros trabajaremos con la base de datos llamada "administracion", que ya hemos creado en el servidor tutorialesprogramacionya.com
Para ver las tablas existentes en una base de datos tipeamos:

show tables;

Al crear una tabla debemos resolver qué campos (columnas) tendrá y que tipo de datos almacenarán cada uno de ellos, es decir, su estructura.
La tabla debe ser definida con un nombre que la identifique y con el cual accederemos a ella.
Creamos una tabla llamada "usuarios", tipeamos:

```
create table usuarios (
        nombre varchar(30),
        clave varchar(10)
);
```

Si intentamos crear una tabla con un nombre ya existente (existe otra tabla con ese nombre), mostrará un mensaje de error indicando que la acción no se realizó porque ya existe una tabla con el mismo nombre.
Para ver las tablas existentes en una base de datos tipeamos nuevamente:

show tables;

Ahora aparece "usuarios" entre otras que ya pueden estar creadas.
Cuando se crea una tabla debemos indicar su nombre y definir sus campos con su tipo de dato. En esta tabla "usuarios" definimos 2 campos:
- **nombre**: que contendrá una cadena de hasta 30 caracteres de longitud, que almacenará el nombre de usuario y
- **clave**: otra cadena de caracteres de 10 de longitud, que guardará la clave de cada usuario.

Cada usuario ocupará un registro de esta tabla, con su respectivo nombre y clave.
Para ver la estructura de una tabla usamos el comando "describe" junto al nombre de la tabla:

describe usuarios;

Aparece lo siguiente:

```
Field  Type         Null
_____________________________
nombre varchar(30)   YES
clave  varchar(10)   YES
```

Esta es la estructura de la tabla "usuarios"; nos muestra cada campo, su tipo, lo que ocupa en bytes y otros datos como la aceptación de valores nulos etc, que veremos más adelante en detalle.
Para eliminar una tabla usamos "drop table". Tipeamos:

drop table usuarios;

Si tipeamos nuevamente:

drop table usuarios;

Aparece un mensaje de error, indicando que no existe, ya que intentamos borrar una tabla inexistente.
Para evitar este mensaje podemos tipear:

drop table if exists usuarios;

En la sentencia precedente especificamos que elimine la tabla "usuarios" si existe.

Tipos de datos

Los tipos de datos que puede haber en un campo, se pueden agrupar en tres grandes grupos:

1. Tipos numéricos
2. Tipos de Fecha
3. Tipos de Cadena

1) Tipos numéricos:

Existen tipos de datos numéricos, que se pueden dividir en dos grandes grupos, los que están en coma flotante (con decimales) y los que no.

TinyInt:

Es un número entero con o sin signo. Con signo el rango de valores válidos va desde -128 a 127. Sin signo, el rango de valores es de 0 a 255

Bit ó Bool:

Un número entero que puede ser 0 ó 1

SmallInt:

Número entero con o sin signo. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535.

MediumInt:

Número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607. Sin signo el rango va desde 0 a16777215.

Integer, Int:

Número entero con o sin signo. Con signo el rango de valores va desde -2147483648 a 2147483647. Sin signo el rango va desde 0 a 429.4967.295

BigInt:

Número entero con o sin signo. Con signo el rango de valores va desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.

Float:

Número pequeño en coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38.

xReal, Double:

Número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308

Decimal, Dec, Numeric:

Número en coma flotante desempaquetado. El número se almacena como una cadena

Tipo de Campo	Tamaño de Almacenamiento
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes

INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 ú 8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D	M+2 bytes sí D > 0, M+1 bytes sí D = 0
NUMERIC(M,D)	M+2 bytes if D > 0, M+1 bytes if D = 0

2) Tipos fecha:

A la hora de almacenar fechas, hay que tener en cuenta que Mysql no comprueba de una manera estricta si una fecha es válida o no. Simplemente comprueba que el mes esta comprendido entre 0 y 12 y que el día esta comprendido entre 0 y 31.

Date:

Tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia

DateTime:

Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos

TimeStamp:

Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037. El formato de almacenamiento depende del tamaño del campo:

Tamaño	Formato
14	AñoMesDiaHoraMinutoSegundo aaaammddhhmmss
12	AñoMesDiaHoraMinutoSegundo aammddhhmmss
8	ñoMesDia aaaammdd
6	AñoMesDia aammdd
4	AñoMes aamm
2	Año aa

Time:

Almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS'

Year:

Almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155. El campo puede tener tamaño dos o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

Tipo de Campo	Tamaño de Almacenamiento

DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

3) Tipos de cadena:

Char(n):

Almacena una cadena de longitud fija. La cadena podrá contener desde 0 a 255 caracteres.

VarChar(n):

Almacena una cadena de longitud variable. La cadena podrá contener desde 0 a 255 caracteres.
Dentro de los tipos de cadena se pueden distinguir otros dos subtipos, los tipo Test y los tipo BLOB (Binary large Object)
La diferencia entre un tipo y otro es el tratamiento que reciben a la hora de realizar ordenamientos y comparaciones. Mientras que el tipo test se ordena sin tener en cuenta las Mayúsculas y las minúsculas, el tipo BLOB se ordena teniéndolas en cuenta.
Los tipos BLOB se utilizan para almacenar datos binarios como pueden ser ficheros.

TinyText y TinyBlob:

Columna con una longitud máxima de 255 caracteres.

Blob y Text:

Un texto con un máximo de 65535 caracteres.

MediumBlob y MediumText:

Un texto con un máximo de 16.777.215 caracteres.

LongBlob y LongText:

Un texto con un máximo de caracteres 4.294.967.295. Hay que tener en cuenta que debido a los protocolos de comunicación los paquetes pueden tener un máximo de 16 Mb.

Enum:

Campo que puede tener un único valor de una lista que se especifica. El tipo Enum acepta hasta 65535 valores distintos

Set:

Un campo que puede contener ninguno, uno ó varios valores de una lista. La lista puede tener un máximo de 64 valores.

Tipo de campo	Tamaño de Almacenamiento
CHAR(n)	n bytes
VARCHAR(n)	n +1 bytes
TINYBLOB, TINYTEXT	Longitud+1 bytes
BLOB, TEXT	Longitud +2 bytes
MEDIUMBLOB, MEDIUMTEXT	Longitud +3 bytes
LONGBLOB, LONGTEXT	Longitud +4 bytes
ENUM('value1','value2',...)	1 ó dos bytes dependiendo del número de valores

SET('value1','value2',...)	1, 2, 3, 4 ó 8 bytes, dependiendo del número de valores

Diferencia de almacenamiento entre los tipos Char y VarChar

Valor	CHAR(4)	Almace namiento	VARCHAR(4)	Almace namiento
''	''	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 byte

Ejercicios:

A) Queremos almacenar los datos de nuestros amigos.

1- Elimine la tabla "agenda" si existe:

```
drop table if exists agenda;
```

2- Cree una tabla llamada "agenda", debe tener los siguientes campos:
nombre varchar(20)
domicilio varchar(30)
y telefono varchar(11)

```
create table agenda(
  nombre varchar(20),
  domicilio varchar(30),
  telefono varchar(11)
);
```

3- Intente crearla nuevamente. Aparece mensaje de error.

4- Visualice las tablas existentes (**show tables**).

5- Visualice la estructura de la tabla "agenda" (**describe agenda**).

7- Intente eliminar la tabla (**drop table if exists agenda**).

B) Queremos almacenar información referente a nuestros libros.

1- Elimine la tabla "libros", si existe.

2- Cree una tabla llamada "libros".
Debe definirse con los siguientes campos:
titulo varchar(20),
autor varchar(30),
y editorial varchar(15)

3- Intente crearla nuevamente. Aparece mensaje de error.

4- Visualice las tablas existentes.

5- Visualice la estructura de la tabla "libros".

6- Elimine la tabla, si existe.

7- Intente eliminar la tabla.