## Servidor de Base de Datos MySQL

Una **base de datos** es un conjunto de datos almacenados que pertenecen a un mismo contexto y que pueden tener relaciones entre sí o no. Para poder manipular una base de datos de forma local en nuestra computadora y poder seguirle el paso a paso de este curso vamos a necesitar un **servidor de base de datos** que permita la instalación de un **SGBD (Sistema Gestor de Base de Datos)** y una serie de herramientas.

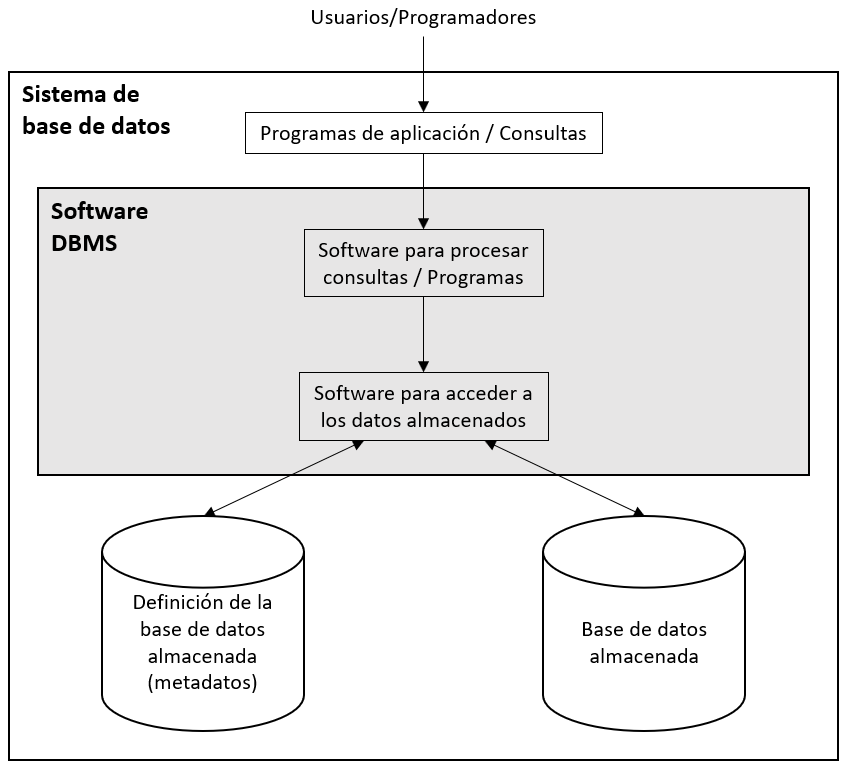
El SGBD que vamos a utilizar es **MySQL**, uno de los más conocidos a nivel mundial y en el ambiente laboral, mientras que como **servidor de base de datos** podemos elegir entre varios dependiendo del sistema operativo con el que contemos; sin embargo, te vamos a dejar dos opciones a continuación: **Wamp Server** (una herramienta EXCLUSIVA para Windows) y **Xampp Server** (Disponible para **Windows, Mac y Distribuciones de Linux**).

## ¿Qué son las bases de datos?: Creando tu primera base de datos

Como vimos anteriormente, «Una ***base de datos*** es un conjunto de datos almacenados que pertenecen a un mismo contexto y que pueden tener relaciones entre si o no.»  Todo el mundo en el que estamos rodeados actualmente se maneja mediante el almacenamiento de datos en diferentes lugares, existiendo bases de datos que son informáticas y otras que no.

**Bases de Datos**

Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. Pueden ser datos cuyo contenido o temática difieren entre sí, pero que poseen relaciones en común.



**SGBD más conocidos**

Dependiendo si la base de datos a tratar es RELACIONAL o NO RELACIONAL, existen diferentes SGBD. Entre ellos se pueden mencionar:

**Relacionales:**

* MySQL
* María DB
* PostgreSQL
* MS Access
* SQL Server
* Entre otros

**No Relacionales:**

* Mongo DB

## Abstracción y Modelado

## La abstracción de datos, por lo general, significa lograr una forma de representación lógica de las situaciones y objetos del mundo real.

## Un modelo de datos proporciona los medios necesarios para conseguir una abstracción.

## Entidades y Atributos

Las **bases de datos** pueden ser representadas de diferentes maneras; sin embargo, por convención internacional existe un tipo de diagrama especial que se utiliza para ello el cual se llama **DER (Diagrama Entidad Relación).**

Todo **DER** tiene una serie de partes que luego podrán verse reflejadas en las tablas que creemos en nuestras bases de datos. Esto se debe a que la tarea de este tipo de diagrama es la de **«abstraer»** elementos de la vida real para que podamos representarlos de **forma lógica** en nuestras bases de datos.

**Modelo de Entidad-Relación**

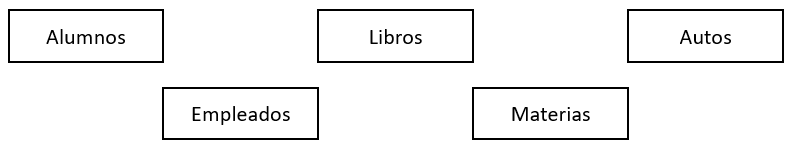
Es un método para diseñar Base de Datos que posteriormente serán implementadas a través de un SGBD. Este modelo se representa a través de diagramas y está formado por varios elementos.

El tipo de Diagrama utilizado para realizar el modelado Entidad-Relación es el DER (Diagrama de Entidad-Relación), el cual pertenece al Lenguaje de modelado universal (UML, Universal Modeling Language).

**DER Componentes**

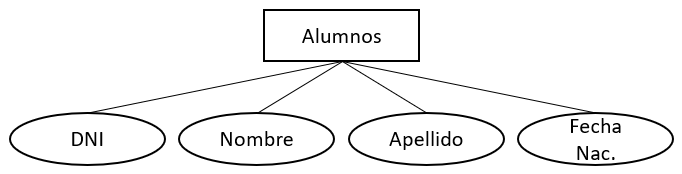
**Entidades**

Las entidades representan cosas u objetos (ya sean reales o abstractos). Se representan en los diagramas como rectángulos. Por ejemplo:



**Atributos**

Los atributos definen o identifican las características propias y por lo general únicas de una entidad. Cada entidad contiene distintos atributos, que dan información sobre la misma. Estos atributos pueden ser de distintos tipos (numéricos, texto, fecha, etc), y se representan por medio de un óvalo o elipse. Por ejemplo:



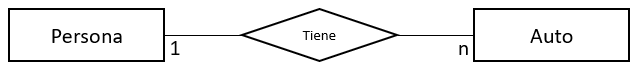
## Relaciones: 1 a 1, 1 a N, y N a N

Las diferentes **entidades** que tenemos en nuestras **bases de datos** no son componentes solitarios, sino que están en constante relación entre sí. Podemos tener tablas o entidades que tal vez traten **elementos totalmente diferentes** entre sí pero que tienen **una relación en común**.

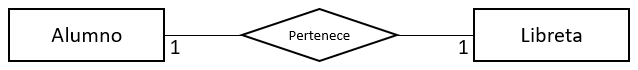
Estas relaciones pueden ser representadas en nuestros **DER (Diagrama Entidad Relación)**, para que luego puedan verse reflejadas en las tablas de nuestra base de datos.

Las relaciones tienen una característica conocida como “cardinalidad”, la cual indica el sentido y la cantidad de “relaciones” existentes entre una entidad y otra. Estas pueden ser:

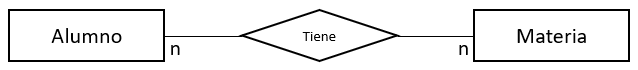
* **1 a N (1 a muchos):** Por ejemplo, una persona puede tener n autos y viceversa, muchos autos pueden ser de una persona.



* **1 a 1:** Por ejemplo. A un alumno pertenece únicamente una libreta y viceversa, una libreta pertenece únicamente a un alumno.



* **N a N (muchos a muchos):** Por ejemplo, muchos alumnos pueden tener muchas materias y viceversa, muchas materias pueden contener a muchos alumnos.

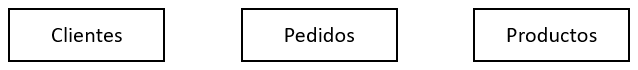


Ejemplo: supongamos que una empresa de venta de electrodomésticos tiene:

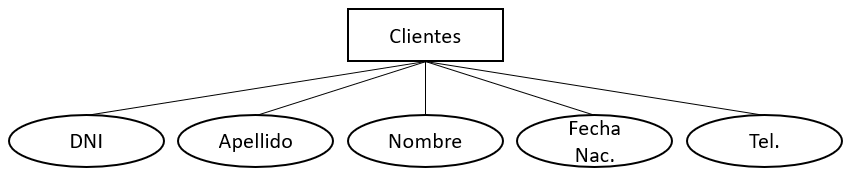
* Clientes
* Pedidos
* Productos.

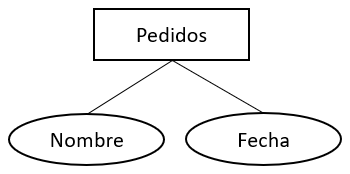
Se desea modelar a través de un DER, la forma en que se implementaría la Base de Datos.

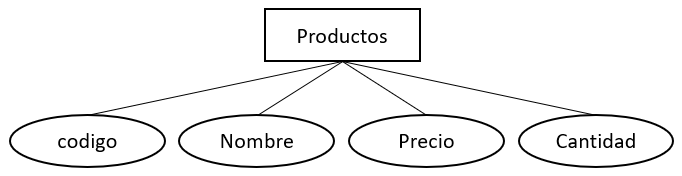
* Primero detectamos las entidades. Como sabemos, eran 3:



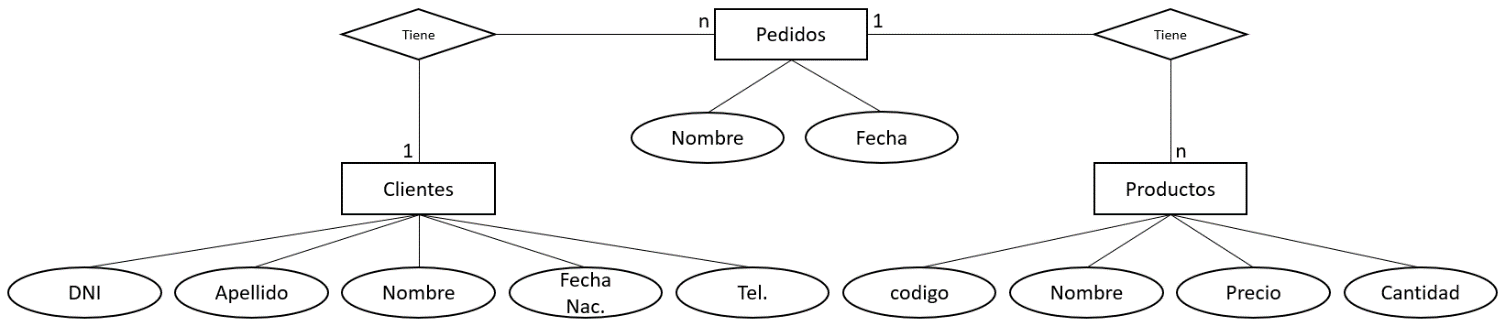
* Ahora detectamos que atributos tienen cada uno de ellos:







* Una vez que conocemos cuales son las entidades y sus atributos, podemos pasar a establecer las relaciones existentes entre sí:



## FOREIGN y PRIMARY Keys (claves foránea y principal)

En la clase anterior vimos que cada una de nuestras entidades o tablas pueden relacionarse entre si teniendo en cuenta una determinada **cardinalidad** que se representa en el **DER**; sin embargo, a la hora de traducir estas relaciones a una base de datos real existen dos fieles amigas que nos van a ayudar a poder hacerlo. Ellas son las **PRIMARY** y **FOREIGN keys**. Las **primary keys** nos permiten identificar de forma única a cada uno de los registros que tengamos almacenados en las tablas de nuestra base de datos, mientras que las **foreign keys** nos van a poder permitir relacionar estas tablas entre si teniendo en cuenta un campo o valor en especial.

Las **claves primarias (Primary Keys)** son valores que identifican de manera única a cada fila o registro de una tabla, esto quiere decir que no se puede repetir. Por ejemplo: un DNI, un código de producto, etc.

Una **clave foránea (Foreign Key)** es un campo de una tabla “X” que sirve para enlazar o relacionar entre sí con otra tabla “Y” en la cual el campo de esta es una llave primaria (Primary Key). Para que sea una clave foránea un campo, esta tiene que ser una llave primaria en otra tabla.

Por ejemplo, en la tabla clientes el dni es una primary key, pero en una tabla “pedidos” representa a quien pertenece ese determinado pedido.

## Consulta SQL (SELECT, INSERT, UPDATE, DELETE)

**SQL (Structured Query Lenguage)** es, como lo dice su nombre, **un lenguaje estructurado para realizar consultas sobre base de datos**. Sirve como idioma «intermediario» entre lo que es el lenguaje natural que hablamos todos los días y un lenguaje que sea capaz de comprender nuestro SGBD.  
En nuestras bases de datos podemos realizar diferentes acciones, las cuales se pueden resumir con las siglas **ABML (Altas, bajas, modificaciones y lecturas)** o **CRUD (en inglés create, read, update y delete)**. Éstas, en las bases de datos relacionales, se llevan a cabo mediante consultas **SQL**.

**Consultas SQL**

Las consultas SQL son los “diálogos” o “preguntas” que se generan entre el usuario y el sistema gestor de base de datos donde se encuentran almacenados ciertos datos.

Existen diferentes sentencias dentro de las consultas SQL las más conocidas son:

***De Lectura:***

* SELECT: Sentencia utilizada para especificar a que atributo (dato) se pretende acceder.
* FROM: Es utilizada en conjunto con el SELECT para especificar desde qué tabla (entidad) se pretende traer el dato.
* WHERE: Sentencia para proponer la condición especifica que debe cumplir el dato que se pretende traer (no es una sentencia obligatoria).

**SELECT dni, apellido, nombre FROM empleados WHERE fecha\_nac > 1990-01-01**

**SELECT \* FROM empleados**

***De Escritura:***

* INSERT INTO: Es utilizada para especificar en que tabla se pretende insertar un dato.
* VALUES: Se utiliza en conjunto con INSERT INTO para especificar que valores irán en la tabla.

**INSERT INTO empleado**

**VALUES (6,'35997425','Romero','Carlos','1985-09-05','2013-03-1','Abogado',99000)**

***De Modificación:***

* UPDATE: Es utilizada para especificar en que tabla se pretende modificar un dato.
* SET: Se utiliza en conjunto con UPDATE para especificar cual será el nuevo valor/dato.
* WHERE: Sentencia para proponer la condición especifica que debe cumplir el dato de referencia del registro que se pretende modificar (no es una sentencia obligatoria, pero si “RECOMENDADA”).

**UPDATE empleado SET nombre = 'Mabel' WHERE num\_legajo = 3**

***De Baja:***

* DELETE FROM: Es utilizada para eliminar un registro de una tabla de forma permanente.
* WHERE: Sentencia para proponer la condición especifica que debe cumplir el dato de referencia del registro que se pretende dar de baja (no es una sentencia obligatoria, pero si “RECOMENDADA”).
* **♫ ♪ No te olvides de poner el WHERE en el DELETE FROM ♪ ♫**

**DELETE FROM empleado WHERE num\_legajo = 3**

**ORDER y GROUP BY**

SQL nos permite llevar a cabo numerosas cantidades de acciones a la hora de obtener los datos que tengamos almacenados en nuestras tablas de cada una de las bases de datos que creemos. Cuando realizamos consultas que nos traen como resultado gran cantidad de datos, es realmente importante que podamos ordenar y agrupar los mismos con la finalidad de tener un mejor acceso o lectura a ellos; para esto, SQL nos ofrece a ORDER BY (para ordenamiento) y GROUP BY (para agrupamiento).

***De Orden y/o Agrupamiento:***

* ORDER BY: Es utilizada para especificar por qué criterio se pretende ordenar los registros de una tabla.
* GROUP BY: Es utilizada para especificar por qué criterio se pretende agrupar a los registros de una tabla.

**SELECT \* FROM empleado ORDER BY apellido ASC**

**SELECT \* FROM empleado ORDER BY sueldo\_neto DESC**

**SELECT cargo FROM empleado GROUP BY cargo**

**COUNT, SUM, MIN, MAX y AVG**

Así como vimos las consultas clásicas que tiene SQL, también existen varias complementarias que nos pueden ayudar a la hora de filtrar, ordenar o mejorar los datos que obtenemos como resultados de cada una de ellas. Para ello, nuestro buen amigo el SELECT se «asocia» con sentencias como COUNT, SUM, MIN, MAX y AVG.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **cod\_prod** | **marca** | **modelo** | **precio** | **stock** |  |
| 775878 | Samsung | S10 | 89999 | 7 |  |
| 986547 | Samsung | S20+ | 123678 | 15 |  |
| 999873 | Xiaomi | Note 9 Pro | 49680 | 3 |  |
| 999874 | Xiaomi | Note 8 | 32600 | 4 |  |
| 2365491 | Iphone | XI Pro | 201000 | 2 |  |

**SELECT COUNT(cod\_prod) FROM productos**

**SELECT MIN(precio) FROM productos**

**SELECT MAX(precio) FROM productos**

**SELECT AVG(precio) FROM productos**

**SELECT SUM(stock) FROM productos**

**SELECT SUM(stock) FROM productos WHERE marca = 'Xiaomi'**

**INNER JOIN**

En la sentencia INNER JOIN sólo se muestran los registros que coinciden tanto de la **tabla A** como de la **tabla B**.

Existen dos formas de darlo a conocer:

1. Forma IMPLÍCITA
2. Forma Explícita

***Explicito***

**SELECT \***

**FROM empleado**

**INNER JOIN departamento**

**ON empleado.id\_departamento = departamento.id\_departamento**

***Implícito***

**SELECT \***

**FROM empleado, departamento**

**WHERE empleado.id\_departamento = departamento.id\_departamento**

**LEFT JOIN, RIGHT JOIN**

***LEFT JOIN:*** Muestra el conjunto completo de los registros de la tabla A, con los registros que coincidan de la tabla B (si es que los hay).

**SELECT \***

**FROM empleado**

**LEFT JOIN departamento**

**ON empleado.id\_departamento = departamento.id\_departamento**

***RIGHT JOIN:*** Es el “reflejo” del Left Join. Muestra el conjunto completo de los registros de la tabla B, con los registros que coincidan de la tabla A (si es que los hay).

**SELECT \***

**FROM empleado**

**RIGHT JOIN departamento**

**ON empleado.id\_departamento = departamento.id\_departamento**

**SUBCONSULTAS**

**¿Qué son las subconsultas?**

Una subconsulta es una consulta SQL que se puede encontrar dentro de otra consulta SQL. Dentro de una sola sentencia SQL se pueden especificar hasta 16 subconsultas y dentro de una subconsulta se pueden especificar otras subconsultas. Las subconsultas se ejecutan de la última a la primera dentro de la sentencia SQL principal en la que aparecen.

Ejemplo:

**SELECT \***

**FROM empleado**

**WHERE sueldo\_neto >=**

**(SELECT AVG(sueldo\_neto)**

**FROM empleado)**

Ejemplo:

**SELECT \***

**FROM hinchas**

**WHERE equipo\_futbol IN**

**(SELECT nombre\_equipo**

**FROM equipos\_primera)**