**Joins y vistas**

**Llaves primarias y foráneas**

Para poder relacionar tablas, necesitamos poder identificar cada uno de los registros de manera única. No podríamos hacer esto usando todos los campos de un registro pues da pie a registros duplicados. Para hacer esto en SQL, se define un concepto llamado llave primaria.

Una [llave primaria](https://es.wikipedia.org/wiki/Clave_primaria) permite identificar de manera única cada uno de los registros de una tabla. Puede usarse un campo que no se repita, por ejemplo, un número de cuenta, un correo electrónico, un código postal, entre otros. Sin embargo, es usual que se añada un campo adicional durante el diseño al que se llama identificador de tipo entero, de esta forma el identificador se incrementa en uno cada que un nuevo registro es añadido.

Para identificar la llave primaria de una tabla podemos aprovechar la instrucción DESCRIBE. Por ejemplo, la siguiente tabla, Persona, incluye un campo id que representa a la llave primaria.

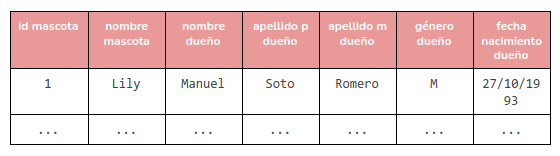
* DESCRIBE Persona;

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Por otro lado, se tienen las [llaves foráneas](https://es.wikipedia.org/wiki/Clave_for%C3%A1nea); este tipo de llaves, representan justamente la relación que pudiera haber entre dos tablas. Por ejemplo, supongamos que se tiene una tabla Mascota y se necesita asociar a cada una de estas con su dueño, es decir, con algún registro dentro de la tabla Persona.

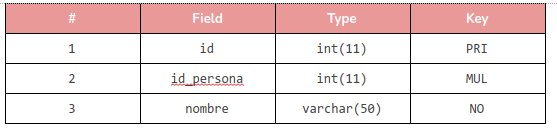
Una forma muy común y que se da usualmente en las hojas de cálculo, es la de manejar toda la información en un sólo registro, por ejemplo:



Esta forma de almacenar la información es útil pues se tienen todos los datos a la mano, sin embargo, si el dueño Manuel, tuviera más de una mascota, se duplicaría su información en varios registros.

La forma en la que SQL evita esta duplicidad de datos es justamente con el uso de llaves, de esta forma, en lugar de almacenar todos los datos de una persona almacenamos únicamente el campo que lo identifica de manera única, es decir, su llave.

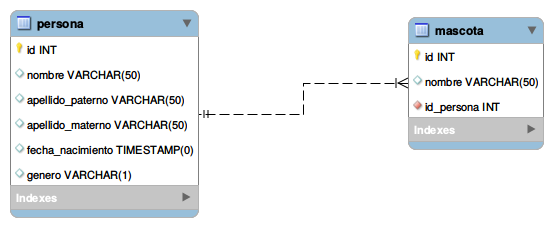
Cuando una tabla almacena la llave primaria de otra tabla, llamamos a esta llave foránea. De esta forma, la tabla Mascota tendría la siguiente descripción.



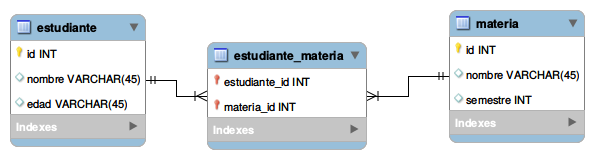
De hecho, una tabla puede tener varios campos como llaves primarias, siempre y cuando, la combinación de sus valores sea única.

**Tipos de relaciones**

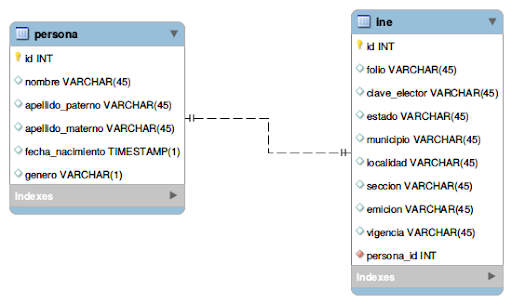
* **Uno a muchos**: Esta forma de relacionar tablas da pie a distintas formas de relacionarlas, por ejemplo, para el caso de las mascotas y su dueño, una mascota tiene un único dueño, pero un dueño (persona) puede tener muchas mascotas. Esto puede apreciarse en la tabla mascota, pues el id de una persona puede aparecer varias veces. Gráficamente esto luce como:



* **Muchos a muchos**: Este tipo de relación permite unir dos tablas que en un principio no tienen relación directa. Por ejemplo, una mascota siempre tiene un dueño, por lo tanto podríamos añadir directamente el identificador dentro de su tabla. ¿Pero qué pasa cuando esto no sucede así?  
  Supongamos ahora, que tenemos una tabla de estudiantes y una tabla de materias. Queremos ahora inscribir a un estudiante dentro de una materia. No podemos relacionarlas agregando una llave foránea pues un estudiante no tiene una materia ni una materia tiene un estudiante como campo.  
  Lo que se hace en este tipo de caso es crear una tabla intermedia que funciona como la unión de dos tablas. Esto luce así:



* **Uno a uno**: Este es el tipo de relación que menos se utiliza, pues quiere decir que ninguna tabla puede repetir registros de la otra. De esta forma, ambas tablas podrían combinarse en una sola. Son creadas con fines muy específicos a partir del contexto en el que fueron creadas, pero son muy poco comunes.   
  Por ejemplo, supongamos que tenemos una tabla de personas y la queremos relacionar con su credencial INE. El INE sólo está asociado a una persona y al mismo tiempo una persona sólo puede tener un INE, por lo tanto, la relación es uno a uno. Esto luce así:

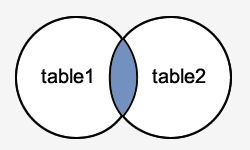


**Joins**

Ahora, usaremos estos conceptos para obtener información de distintas tablas, mediante lo que se conoce como [join](https://es.wikipedia.org/wiki/Join) que se traduce en ocasiones como reunión. En MySQL tenemos tres tipos de join:

* **INNER JOIN** (o simplemente JOIN)
* **LEFT OUTER JOIN** (o simplemente LEFT JOIN)
* **RIGHT OUTER JOIN** (o simplemente RIGHT JOIN)

**JOIN**: Un join, relaciona dos tablas, trayendo todos los campos de éstas siempre y cuando se cumpla la condición de relación. Visualmente un join, luce así:

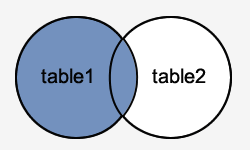


Por ejemplo, la consulta:

* SELECT \*  
  FROM persona  
  JOIN mascota  
   ON persona.id = mascota.id\_persona;

Obtiene una tabla con todos los campos de la tabla persona, todos los campos de la tabla mascota y agrega únicamente aquellos registros donde el id de la persona sea igual al id de la persona asociada a la mascota.

**LEFT JOIN**: Un join izquierdo, relaciona dos tablas siempre y cuando se cumpla la condición de relación, sin embargo, traerá todos los registros de la tabla izquierda y únicamente aquellos registros que cumplan con la condición de relación de la otra tabla. Visualmente un join izquierdo, luce así:

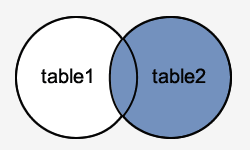


Por ejemplo, la consulta:

* SELECT \*  
  FROM persona  
  LEFT JOIN mascota  
   ON persona.id = mascota.id\_persona;

Obtiene una tabla con todos los registros de la tabla persona, pero sólo aquellos de la tabla mascota en donde el id de persona sea igual.

**RIGHT JOIN**: Un join derecho es inverso al join izquierdo, relaciona dos tablas siempre y cuando se cumpla la condición de relación, sin embargo, traerá todos los registros de la tabla derecha y únicamente aquellos registros que cumplan con la condición de relación de la otra tabla. Visualmente un join derecho, luce así:



Por ejemplo, la consulta:

* SELECT \*  
  FROM persona  
  RIGHT JOIN mascota  
   ON persona.id = mascota.id\_persona;

Obtiene una tabla con todos los registros de la tabla mascota, pero sólo aquellos de la tabla persona en donde el id de persona sea igual.

**Vistas**

A diferencia de una tabla, una [vista](https://es.wikipedia.org/wiki/Vista_(base_de_datos)) es la representación virtual de una consulta en formato de tabla o, dicho de otro modo, es el resultado de guardar una consulta para poder consultarla como a cualquier otra tabla. Son útiles cuando el tiempo de procesamiento de una consulta es alto y por lo tanto no tenemos que ejecutarla una y otra vez pues estará disponible.

Para crear una vista, basta con darle un nombre a la consulta que estamos tomando como base, por ejemplo, la siguiente vista toma la consulta que asocia mascotas con su dueño, llamamos a esta vista, *duenios*.

* CREATE VIEW duenios AS  
   SELECT \*  
   FROM persona  
   JOIN mascota  
   ON persona.id = mascota.id\_persona;

Una vez creada, podemos consultar los campos que arroja la consulta asociada a una vista, pero en lugar de llamar a la consulta, llamamos a la vista con el nombre que se le dio.

* SELECT \*  
  FROM duenios  
  WHERE id\_persona = 2;