



CICLO 2

[FORMACIÓN POR CICLOS]

Introducción a **MYSQL**



Ingeni@
Soluciones TIC



UNIVERSIDAD
DE ANTIOQUIA
Facultad de Ingeniería

Bases de Datos Relacionales

Las bases de datos nos permiten almacenar y gestionar información de manera organizada y siguiendo una estructura determinada. En las bases de datos relacionales, dicha estructura está definida en forma de **tablas**, compuestas por filas y columnas, las cuales guardan datos relacionados bajo un concepto común.

Las columnas corresponden a los **campos**, y se pueden derivar de los atributos de una entidad de un diagrama entidad-relación existente. Cada campo representa un dato específico con el que cuentan todas las instancias. Ejemplos de campos son: el nombre, el documento de identidad de una persona, o bien la placa de un auto o su color.

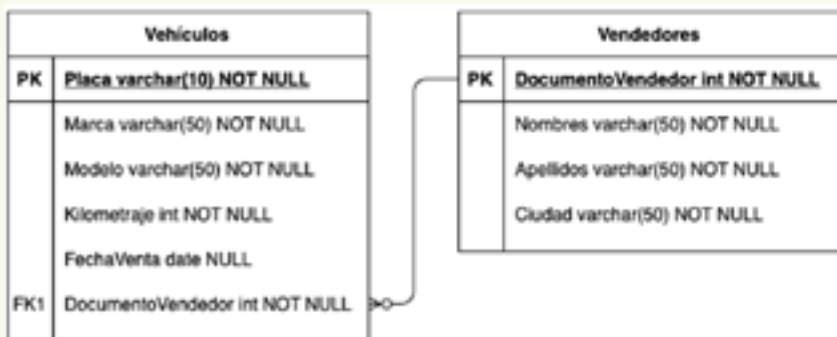
Las filas, por su parte, corresponden a los **registros**, representando cada registro una instancia de una entidad, y por tanto los valores de todos sus campos. Un ejemplo de un registro serían todos los datos dentro de una tabla correspondientes al vehículo con la placa "ACD 427", tal como lo podemos observar a continuación.

Tabla: Vehículos					
Placa	Marca	Modelo	Kilometraje	FechaVenta	DocumentoVendedor
ACD 427	Renault	Stepway	25000	05-7-2020	1964
IYU 521	Renault	Clío	70200	<i>null</i>	2853
POI 700	Nissan	March	45345	<i>null</i>	1964

Arriba podemos apreciar la tabla Vehículos, la cual cuenta con seis campos y tres registros. Los campos son: la placa, la marca, el modelo, el kilometraje, la fecha de venta y el documento del vendedor del vehículo. Los registros corresponden a tres vehículos diferentes con placas ACD 427, IYU 521 y POI 700. Al menos uno de los campos de una tabla debería ser su clave principal. En el caso de arriba es la placa (lo cual indicamos subrayándola). Esto quiere decir, que dos vehículos diferentes no pueden tener la misma placa, y que esta es obligatoria. Además, cada campo tiene diferentes características, entre ellas el tipo de dato. Arriba podemos intuir que los campos Placa, Marca y Modelo son alfanuméricos (incluyen letras y números), los campos Kilometraje y DocumentoVendedor son numéricos, y el campo FechaVenta tendrá como tipo de dato Fecha. Finalmente, podemos observar que algunos registros tienen su fecha con valor en *null*. Eso quiere decir que esos vehículos no registran fecha, y al tratarse la fecha de un campo opcional, registran ese valor, en lugar de una fecha como tal. A continuación, tenemos la tabla Vendedores, con sus propios datos.

Tabla: Vendedores			
<u>DocumentoVendedor</u>	Nombres	Apellidos	Ciudad
1964	Juan	Rodríguez	Medellín
2853	José	González	Bogotá
6501	Luis	Gómez	Cali

La tabla Vendedores cuenta con cuatro campos: el documento del vendedor, sus nombres, apellidos y ciudad. De estos, el documento es numérico y los demás alfanuméricos (letras y números). Además, su clave principal es DocumentoVendedor, lo que implica que dos vendedores cualquiera deberán tener documento (no podrá ser *null*) y para todos los vendedores dicho documento debe ser diferente. Si nos fijamos, podremos observar que ambas tablas comparten el campo DocumentoVendedor. Esto es porque por medio de este campo se da la relación entre las tablas Vehículos y Vendedores. Esto nos da la posibilidad de que un vendedor tenga asignados varios vehículos, constituyéndose DocumentoVendedor como clave foránea en la tabla Vehículos, y siendo esto equivalente al diagrama entidad-relación que se observa a continuación.



SQL y los SGBD

Pero ¿Cómo se construye y gestiona todo lo anterior? Pues bien, las bases de datos relacionales se construyen y consultan usando el lenguaje SQL. **SQL** (*Structured Query Language*) es un lenguaje cuyo propósito es la gestión de datos relacionales, almacenadas en un sistema de gestión de bases de datos (SGBD). Un **SGBD**, es un sistema de software que administra la forma en que los datos son almacenados, mantenidos y recuperados. Entre ellos, contamos con MySQL, Oracle, MS SQL Server o PostgreSQL, cada uno de los cuales soporta el lenguaje estándar SQL con algunos elementos adicionales propios de cada uno.

¿Qué es MySQL?

Uno de los SGBD más populares en los últimos años es MySQL. **MySQL** está orientado a la gestión de bases de datos relacionales, principalmente para aplicaciones web. Inicialmente, MySQL era propiedad de MySQL AB, empresa que fue adquirida por Sun Microsystems, y esta a su vez, fue adquirida por Oracle. Lo anterior llevó a que el código fuente de MySQL dejase de ser libre, dando lugar a diferentes *forks*, como MariaDB.

Al ser ampliamente utilizado (incluso por empresas como Google, Facebook o Wikipedia), MySQL puede ser desplegado en prácticamente cualquier plataforma y accedido desde cualquier lenguaje de programación (incluidos Python y Java). Finalmente, cabe mencionar que MySQL soporta el lenguaje SQL estándar con algunos elementos de sintaxis adicionales. Las características de MySQL lo hacen un SGBD simple, rápido y fácil de usar.

¿Cómo instalar MySQL?

MySQL tiene una arquitectura cliente-servidor. Es decir, El SGBD como tal se ejecuta permanentemente (o bajo demanda) a la espera de conexiones y peticiones. Por otro lado, nosotros nos conectamos al mismo desde un cliente, ya sea una aplicación diseñada para tal fin (como MySQL Workbench) o nuestra propia aplicación desarrollada usando Java. Para lograr lo anterior primero será necesario instalar ambas cosas:

- En primer lugar, es necesario descargar e instalar la versión de SQL apropiada para su equipo y necesidades: <https://dev.mysql.com/downloads/mysql/>
- Posteriormente, será necesario instalar una aplicación que facilite la gestión del servidor, como puede ser MySQL Workbench: <https://www.mysql.com/products/workbench/>
- Sin embargo, podemos instalar XAMPP, en cuyo caso se instalarán al mismo tiempo MariaDB (*fork* de MySQL), un servidor Apache y phpMyAdmin, esta última una herramienta basada en la web para la gestión de MySQL y MariaDB: <https://www.apachefriends.org/es/download.html>

Uso de MySQL

Nosotros interactuaremos con una base de datos de MySQL mediante consultas. Estas consultas nos permitirán crear tablas, obtener datos de ellas, insertar registros, modificarlos, eliminarlos, entre muchas otras cosas más. A continuación, haremos un breve recuento de las consultas básicas de MySQL, implementando las tablas descritas arriba.

Creación de bases de datos

Antes de crear cualquier tabla y de realizar consultas sobre ella, es necesario crear una base de datos. Esto se logra mediante la consulta `CREATE DATABASE`. Por ejemplo, a continuación, creamos una base de datos llamada concesionario, donde guardaremos las tablas definidas en los ejemplos de arriba.

```
CREATE DATABASE concesionario;
```

Creación de tablas

Para crear una tabla, usaremos la consulta `CREATE TABLE`. Para esto, debemos seguir la sintaxis definida por el lenguaje, tal como podemos verlo a continuación.

```
CREATE TABLE Vehiculos (  
  placa VARCHAR(10) NOT NULL,  
  marca VARCHAR(50) NOT NULL,  
  modelo VARCHAR(50) NOT NULL,  
  kilometraje INT NOT NULL,  
  fecha_venta DATE,  
  documento_vendedor INT NOT NULL,  
  PRIMARY KEY ( placa )  
);  
  
CREATE TABLE Vendedores (  
  documento_vendedor INT NOT NULL,  
  nombres VARCHAR(50) NOT NULL,  
  apellidos VARCHAR(50) NOT NULL,  
  ciudad VARCHAR(50) NOT NULL,  
  PRIMARY KEY ( documento_vendedor )  
);
```

En el recuadro anterior podemos apreciar como se crean las dos tablas, `vehiculos` y `vendedores`, usando dos consultas SQL. En cada una se define el nombre de la tabla y entre paréntesis los diferentes campos, incluyendo su nombre, tipo de dato y si son obligatorios (`NOT NULL`) o no. Finalmente, con la cláusula `PRIMARY KEY` indicamos cuál de los campos de la tabla es la clave primaria.

Alteración de propiedades de una tabla

Es posible alterar una tabla, así como sus propiedades o las propiedades de sus campos. Esto se puede lograr mediante la cláusula `ALTER TABLE`. Arriba, creamos las tablas y definimos sus claves primarias, pero olvidamos definir sus claves foráneas, para lo cual podremos utilizar después `ALTER TABLE`, así.

```
ALTER TABLE Vehiculos
ADD FOREIGN KEY (documento_vendedor)
REFERENCES Vendedores (documento_vendedor);
```


Con la anterior consulta estamos definiendo el campo `documento_vendedor` de la tabla `vehiculos` como clave foránea, vinculada con la clave primaria `documento_vendedor` de la tabla `vendedores`. Y así, implementamos la relación entre las dos tablas que se puede observar en el diagrama entidad relación que se muestra más arriba.

Inserción de registros

Una vez se han creado las tablas, podremos insertar registros en ellas. Para esto se utiliza la consulta `INSERT INTO`. Obviamente, debemos tener en cuenta las restricciones y tipos de datos de los diferentes campos. Es así como, a continuación, se insertan registros en las dos tablas mostradas arriba.

```
INSERT INTO Vendedores (documento_vendedor,
nombres, apellidos, ciudad)
VALUES ('1964', 'Juan', 'Rodríguez', 'Medellín');
INSERT INTO Vendedores (documento_vendedor,
nombres, apellidos, ciudad)
VALUES ('2853', 'José', 'González', 'Bogotá');

INSERT INTO Vehiculos (placa, marca, modelo,
kilometraje, fecha_venta, documento_vendedor)
VALUES ('ACD 427', 'Renault', 'Stepway', '25000',
'2020-7-05', '1964');
INSERT INTO Vehiculos (placa, marca, modelo,
kilometraje, fecha_venta, documento_vendedor)
VALUES ('IYU 521', 'Renault', 'Clío', '70200',
null, '2853');
```



El código anterior incluye cuatro consultas. Con las dos primeras insertamos dos registros en la tabla `vendedores`, creada arriba. Con las dos últimas, insertamos dos registros en la tabla `vehiculos`. Estos registros corresponden a las tablas de ejemplo que se pueden apreciar más arriba. Las consultas `INSERT INTO`, en este caso, incluyen:

- El nombre de la tabla donde insertaremos el registro.
- Entre paréntesis, y separados por comas, los nombres de los campos donde registraremos los datos.
- Y luego de la palabra `VALUES`, y entre paréntesis y separados por comas, los valores que vamos a asignar a los campos respectivos. Si nos fijamos, estos corresponden exactamente entre ellos.

La consulta `INSERT INTO` tiene diferentes variantes. Sin embargo, siempre será necesario tener en cuenta el número y tipo de los campos, así como las restricciones de estos.

Consulta o recuperación de registros

Una vez tenemos una o varias tablas y estas cuentan con registros, podemos realizar consultas para recuperar uno o varios registros que cumplan determinada condición, usando la cláusula `SELECT`. Estas consultas siguen, por lo general, la siguiente forma:

```
SELECT campo1, campo2, ... FROM nombre_tabla
WHERE condición;
```


Podemos observar que en la consulta se especifican:

- Después de la cláusula `SELECT`, la lista de campos de los cuales queremos obtener la información con la consulta.
- Después de la palabra `FROM`, la tabla desde la cual queremos obtener la información.
- Y siguiendo a `WHERE`, la o las condiciones que deben cumplir cada uno de los registros de los cuales queremos obtener información.

Dicho lo anterior, a continuación, tenemos algunos ejemplos.

```
SELECT * FROM vendedores;
```

En el ejemplo anterior, en lugar de la lista de campos, tenemos un asterisco (*), lo cual implica que se quieren recuperar todos los campos de cada registro. Luego, al no incluir una condición, recupera todos los registros de la tabla.




```
SELECT placa, marca, fecha_venta FROM
vehiculos;
```

En este ejemplo, al no incluir una condición, se recuperan todos los registros de la tabla vehículos, pero sólo los campos placa, marca y fecha_venta.

```
SELECT documento_vendedor, nombres, apellidos,
ciudad FROM vendedores WHERE ciudad =
'Medellín';
```

En el ejemplo de arriba, se consultan los campos documento_vendedor, nombres, apellidos y ciudad de la tabla vendedores en los cuales ciudad sea Medellín. Si ningún registro cumple la condición, no devolverá ninguno.

```
SELECT placa, modelo, kilometraje FROM
vehiculos WHERE fecha_venta > '2020-1-01';
```

En este ejemplo, se recuperan los campos placa, modelo y kilometraje de los registros de la tabla vehículos cuya fecha de venta sea superior al primero de enero de 2020.

```
SELECT placa, marca, modelo, kilometraje
FROM vehículos WHERE kilometraje > '20000' AND
modelo = 'Stepway';
```

En este ejemplo, se recuperan los campos placa, marca, modelo y kilometraje de los vehículos cuyo kilometraje sea mayor a 20000 y sean del modelo Stepway. Como podemos observar en este caso, las condiciones pueden ser compuestas, y existe una notación que permite una amplia variedad de notaciones en ese sentido.

Actualización de registros

Es posible que necesitemos actualizar, o bien, modificar uno o varios registros existentes en una tabla. Para eso, usamos la cláusula UPDATE. Estas consultas siguen, por lo general, la siguiente forma:

```
UPDATE nombre_tabla
SET columna1 = valor1, columna2 = valor2, ...
WHERE condición;
```

En una consulta de esta forma se especifican:

- Después de la palabra `UPDATE`, el nombre de la tabla en la cual queremos actualizar o modificar información.
- Después de la palabra `SET` y separados por comas, los cambios que queremos realizar, asignando el nuevo valor a su respectivo campo.
- Después de `WHERE`, tal como en las consultas `SELECT`, la o las condiciones que deben cumplir los registros en los que actualizaremos los campos especificados.

Lo anterior, lo podemos observar en los siguientes ejemplos:

```
UPDATE vehiculos
SET kilometraje = '100000'
WHERE placa = 'ACD 427';
```

En el primer ejemplo buscamos actualizar la tabla `vehículos`, cambiando el campo `kilometraje` a 100000 en el registro cuya placa es ACD 427. En este caso estaremos cambiando el kilometraje de un solo vehículo, ya que la placa es clave principal.

```
UPDATE vendedores
SET ciudad = 'Medellín';
```

En este ejemplo, al no incluir una condición, estamos cambiando la ciudad de todos los registros de la tabla `vendedores` a Medellín.

```
UPDATE vehiculos
SET modelo = 'Sanderó', fecha_venta = '2021-5-06'
WHERE documento_vendedor = '2853';
```

En el ejemplo anterior, se cambia el valor de los campos `modelo` y `fecha_venta` a el o los vehículos asignados al vendedor con documento 2853.

```
UPDATE vehiculos
SET kilometraje = '20000'
WHERE fecha_venta >= '2021-1-01' OR fecha_venta < '2020-1-01';
```

En este último ejemplo, en cambio, actualizamos a 20000 el kilometraje de los vehículos cuya fecha de venta no está en el año 2020. Como podemos observar, en este caso también es posible hacer diferentes combinaciones lógicas en la cláusula `WHERE`.

Eliminación de registros

Entre otras consultas, en SQL en general, y MySQL en particular, contamos con aquellas usadas para la eliminación de registros. Para eso, usamos la cláusula `DELETE`. Estas consultas siguen, por lo general, la siguiente forma:

```
DELETE FROM nombre_tabla WHERE condición;
```

Basta con incluir:

- Después de `DELETE FROM`, el nombre de la tabla en la cual eliminaremos los registros.
- Luego de la cláusula `WHERE`, como siempre, la condición que deberán cumplir los registros a eliminar.

A continuación, podemos observar tres ejemplos:

```
DELETE FROM vehiculos;
```

En el primer ejemplo, al carecer de una condición, con la consulta se busca eliminar todos los registros de la tabla `vehículos`.

```
DELETE FROM vendedores WHERE ciudad = 'Bogotá';
```

En el segundo ejemplo, se busca eliminar todos los registros de la tabla `vendedores` asociados a la ciudad de Bogotá.

```
DELETE FROM vehiculos WHERE marca = 'Renault'  
AND fecha_venta > '2020-1-01';
```

Finalmente, en el tercer ejemplo, se busca eliminar todos los registros de la tabla `vehículos`, cuya marca sea Renault y su fecha de venta sea a partir del 1 de enero de 2020.

Elementos Adicionales del Lenguaje

SQL en general y MySQL en particular tienen notaciones muy amplias, con una gran cantidad de elementos imposibles de abarcar en el presente módulo. Para su estudio, se recomienda consultar la documentación oficial del lenguaje¹, teniendo siempre en cuenta la versión que se quiere utilizar.

¹Documentación oficial de MySQL: <https://dev.mysql.com/doc/>