# MySQL

## Introducción

En programación es prácticamente inevitable trabajar con algún tipo de sistema de gestión de bases de datos. Cualquier programa que imaginemos tarde o temprano necesitará almacenar datos en algún lugar, como mínimo para poder almacenar la lista de usuarios autorizados, sus permisos y propiedades.

MySQL es el sistema de gestión de bases de datos relacional más extendido en la actualidad al estar basado en código abierto. Desarrollado originalmente por MySQL AB, fue adquirida por Sun MicroSystems en 2008 y esta su vez comprada por Oracle Corporation en 2010, la cual ya era dueña de un motor propio InnoDB para MySQL.

## Bases de datos relacionales

Una base de datos relacional es una colección de información que organiza datos en relaciones predefinidas, en la que los datos se almacenan en una o más tablas (o "relaciones") de columnas y filas, lo que facilita su visualización y la comprensión de cómo se relacionan las diferentes estructuras de datos entre sí. Las relaciones son conexiones lógicas entre las diferentes tablas y se establecen a partir de la interacción entre ellas.

### La definición

Una base de datos relacional (RDB) es una forma de estructurar información en tablas, filas y columnas. Un RDB tiene la capacidad de establecer vínculos (o relaciones) entre información mediante la unión de tablas, lo que facilita la comprensión y la obtención de estadísticas sobre la relación entre varios datos.

### Un poco de historia y el modelo de base de datos relacionales

Desarrollado por EF Codd desde IBM en la década de 1970, el modelo de base de datos relacional permite que cualquier tabla se relacione con otra mediante un atributo común. En lugar de usar estructuras jerárquicas para organizar los datos, Codd propuso un cambio a un modelo de datos en el que los datos se almacenan, se consultan y se relacionan en tablas sin reorganizar las tablas que los contienen.

Considera la base de datos relacional como una colección de archivos de hojas de cálculo que ayudan a las empresas a organizar, administrar y relacionar datos. En el modelo de base de

datos relacional, cada “hoja de cálculo” es una tabla que almacena información, representada como columnas (atributos) y filas (registros o tuplas).

Los atributos (columnas) especifican un tipo de datos, y cada registro (o fila) contiene el valor de ese tipo de datos específico. Todas las tablas de una base de datos relacional tienen un atributo conocido como clave primaria, que es un identificador único de una fila, y cada fila se puede usar para crear una relación entre tablas diferentes mediante una clave externa (una referencia a una clave primaria de otra tabla existente).

## Crear un contenedor de MySQL usando Docker

Para crear una instancia de base de datos de MySQL en Docker podemos hacerlo de dos formas, por línea de comandos con Docker propiamente dicho o creando un archivo docker-compose.yml que nos agilice el trámite. Veamos ambas formas:

### Con comandos en Docker

Escribimos en nuestra terminal el siguiente comando (recordamos tener Docker corriendo):

```
docker run --name my-mysql-instance -p 3306:3306 -e
MYSQL_ROOT_PASSWORD=root -d mysql:latest
```

### Utilizando Docker Compose

Debemos crear en alguna carpeta que elijamos un archivo docker-compose.yml con el siguiente contenido:

```yaml
version: '2'
services:
  mysql:
    image: mysql
    ports:
      - '3306:3306'
    environment:
      - MYSQL_ROOT_PASSWORD=root
```

Luego simplemente nos posicionamos en donde hemos creado este archivo y ejecutamos el comando:

```
docker compose up
```

## Cómo comenzar a trabajar con bases de datos

Una vez tengamos nuestra instancia corriendo, debemos tener alguna herramienta para conectarnos a dicha instancia y poder trabajar.

Existen infinidad de aplicaciones para esto, entre ellas podemos encontrar:

- JetBrains DataGrip (la que yo uso), es una tool Paga (https://www.jetbrains.com/datagrip/).
- MySQL WorkBench, una aplicación gratuita distribuida por MySQL (https://www.mysql.com/products/workbench/).

Ahora que ya tenemos nuestra base de datos corriendo y nuestra tool para conectarnos debemos utilizar el siguiente “connection string” (así se denomina al protocolo que utilizamos para conectarnos a una base de datos, esto se utilizará en nuestras aplicaciones y todas aquellas que se quieran conectar a dicha instancia).

```
server=localhost;port=3306;user=root;password=root;Persist Security
Info=False;
```

## Crear una base de datos y tablas

Ya tenemos todo listo para comenzar a crear tablas y datos, pero antes que eso necesitamos crear una base de datos o esquema, aquí es donde vivirán todas nuestras tablas para una aplicación en particular, si poseemos más de una aplicación es recomendable tener un esquema por cada una de ellas.

Para crear un esquema utilizamos el siguiente script:

```
create schema prueba_db;
```

Para crear una tabla en nuestro nuevo esquema podemos utilizar un script como el que sigue:

```
create table prueba_db.tabla1
(
    id     int auto_increment,
    nombre VARCHAR(100)           null,
    fecha  datetime default NOW() not null,
    constraint tabla1_pk
```

```
        primary key (id)
);
```

Aquí un poco de detalle…

- "*id int auto_increment*" : significa que la columna se llama *id* tiene un tipo *int* y es auto incremental (es decir, cada vez que insertemos un registro el valor irá en aumento), y al ser auto incremental es *no nullable* (no puede tener valor vacío).
- "*nombre VARCHAR(100) null*": significa que la columna se llama *nombre* posee un tipo *varchar(100)*, es decir acepta alfanuméricos de hasta 100 caracteres y *null* significa que el valor puede ser vacío.
- "*fecha  datetime default NOW() not null*": la tercer columna se llama *fecha*, posee un tipo *datetime* (fecha y hora), *default now()* significa que si no se provee un valor asignará la función *NOW()* es decir, la fecha y hora del servidor de base de datos y *not null* significa que no puede contener valores vacíos.
- La última parte es un poco difícil, aquí se setea una "*constraint*" es decir una restricción, existen muchos tipos de *constraints*, *primary key* es una de ellas… para mantenerlo simple, una clave primaria (*primary key*), es el valor que utilizaremos nosotros y el motor de base de datos para armar referencias y para realizar búsquedas. Por ende esta clave no puede ser duplicada.
  Podemos usar cualquier columna como clave primaria, para este ejemplo utilizaremos la columna *id*.

En caso de que queramos borrar dicha tabla, podemos utilizar el siguiente script:

```
drop table prueba_db.tabla1;
```

Para la siguiente sección vamos a armar un ejemplo concreto, imaginemos un sistema de compras (ordenes de compra y clientes), obviamente va a ser muy simple, tendremos dos tablas:
Customer: la cual tendrá una información básica de los clientes
Orders: la cual tendrá la información de las órdenes realizadas por los clientes (obviamente con su referencia).

Customer Table:

```
create table prueba_db.Customer
(
    id               int auto_increment,
    billing_address  varchar(250) not null,
    shipping_address VARCHAR(250) not null,
    name             varchar(150) not null,
    constraint Customer_pk
        primary key (id)
```

```
);
```

Orders Table (prestar atención en la constraint fk, significa Foreign Key):

```
create table prueba_db.`Order`
(
    id            int auto_increment,
    customer_id   int                    not null,
    order_date    datetime default now() not null,
    shipping_date datetime               null,
    status        varchar(50)            not null,
    constraint Order_pk
        primary key (id),
    constraint Order_Customer_id_fk
        foreign key (customer_id) references prueba_db.Customer (id)
);
```

## Importar datos de un archivo .sql

Desde la terminal podemos importar directamente un archivo sql siguiendo el siguiente formato:

```
mysql -uusuario -p tabla < archivo.sql
```

Reemplazamos “usuario” por nuestro usuario, “tabla” por la tabla que queremos afectar y “archivo.sql” por el archivo sql que deseamos importar. Es importante tener en cuenta que deberíamos tener creada la tabla (aunque esté vacía) antes de hacer la importación.

## SELECT

SELECT / seleccionar / obtener / retrieve, son las distintas palabras que utilizamos para este comando, el cual utilizamos básicamente para obtener un conjunto de registros.

Ejemplos:

- Supongamos que queremos tener un listado de Clientes:

```
SELECT * FROM prueba_db.Customer;
```

- Supongamos que queremos obtener un listado de órdenes que se encuentren en estado “DESPACHADO”.

```
SELECT * FROM prueba_db.`Order` WHERE status = 'DESPACHADO';
```

- Ahora un poco más complejo, supongamos que queremos obtener el listado de órdenes con el nombre del cliente que ha hecho el pedido.

```
SELECT o.id as order_id, o.order_date, o.shipping_date, o.status as
order_status, c.id as customer_id, c.name as customer_name
  FROM prueba_db.`Order` o, prueba_db.Customer c
 WHERE o.customer_id = c.id
       AND o.status = 'DESPACHADO';
```

Veamos que el ejemplo es mucho más complejo, ya que cruzamos dos tablas utilizando la clave foránea customer_id. Además hemos utilizado alias para las columnas y para las tablas.

## INSERT

La sentencia INSERT se utiliza para agregar registros a las tablas.

Ejemplos:

- Agregar un registro a la tabla Customer, prestar atención que no se detalla la PK (columna id), ya que es autogenerada.

```
INSERT INTO prueba_db.Customer (billing_address, shipping_address, name)
VALUES ('Calle 1, numero 10, CABA, Argentina', 'Calle 1, numero 10, CABA,
Argentina', 'John Doe');
```

## UPDATE

La sentencia UPDATE se utiliza obviamente para actualizar registros existentes en las distintas tablas. Es importante recalcar que si no ponemos una cláusula WHERE, podemos actualizar todos los registros de dicha tabla.

Ejemplo:

- Actualizaremos el estado de la órden con id = 5, de DESPACHADO a ENTREGADO.

```
UPDATE prueba_db.`Order` SET status = 'ENTREGADO' WHERE id = 5;
```

## DELETE

La sentencia DELETE claramente se utiliza para borrar registros, aquí también es muy importante no olvidar la cláusula WHERE ya que podemos borrar todos los registros (me ha pasado 🤦 ).

Ejemplo:

- Borrar el cliente con id = 4

```
DELETE FROM prueba_db.Customer WHERE id = 4;
```

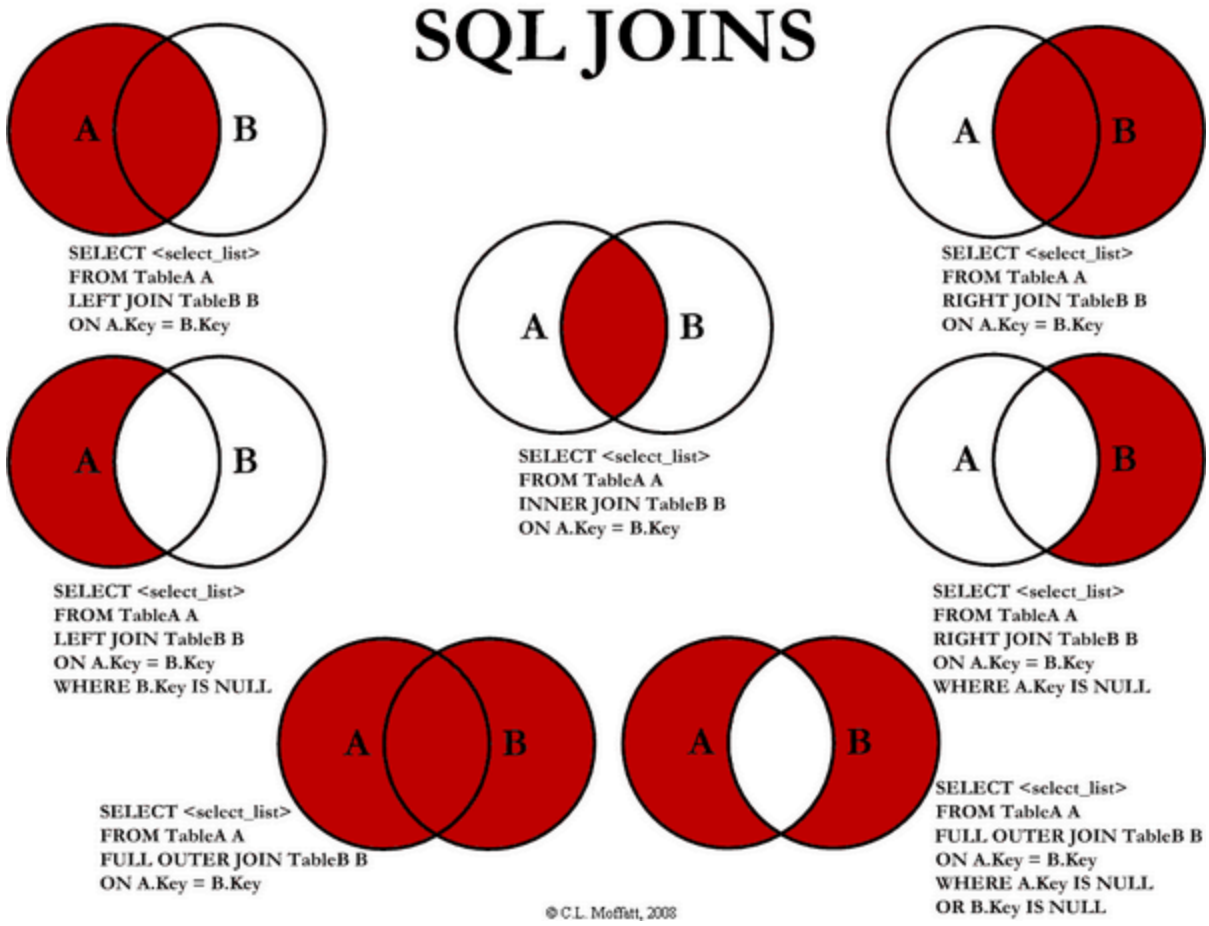
- Borrar las órdenes que se encuentren en estado RECHAZADO

```
DELETE FROM prueba_db.`Order` WHERE status = 'RECHAZADO';
```

## JOIN

Existen varios tipos de JOIN, se utiliza para unir tablas, es decir obtener registros de distintas tablas unidos por alguna referencia. Recordar teoría de conjuntos.

- INNER JOIN, es la intersección entre dos grupos.
- LEFT JOIN, son todos los de la IZQUIERDA (es decir los que ponemos como primer tabla), y los de la intersección.
- RIGHT JOIN, son todos los de la DERECHA (es decir, segunda tabla), mas los de la intersección.




Ejemplo:

- El tercer ejemplo de SELECT podríamos escribirlo de la siguiente forma:

```
SELECT o.id as order_id, o.order_date, o.shipping_date, o.status as
order_status, c.id as customer_id, c.name as customer_name
  FROM prueba_db.`Order` o
 INNER JOIN prueba_db.Customer c
    ON o.customer_id = c.id
 WHERE o.status = 'DESPACHADO';
```

## Índices

Un índice es un puntero a una fila de una determinada tabla de nuestra base de datos. Pero… ¿Qué significa exactamente un puntero? Pues bien, un puntero no es más que una referencia que asocia el valor de una determinada columna (o el conjunto de valores de una serie de

columnas) con las filas que contienen ese valor (o valores) en las columnas que componen el puntero.

Los índices mejoran el tiempo de recuperación de los datos en las consultas realizadas contra nuestra base de datos. Pero los índices no son todo ventajas, la creación de índices implica un aumento en el tiempo de ejecución sobre aquellas consultas de inserción, actualización y eliminación realizadas sobre los datos afectados por el índice (ya que tendrán que actualizarlo). Del mismo modo, los índices necesitan un espacio para almacenarse, por lo que también tienen un coste adicional en forma de espacio en disco.

La construcción de los índices es el primer paso para realizar optimizaciones en las consultas realizadas contra nuestra base de datos. Por ello, es importante conocer bien su funcionamiento y los efectos colaterales que pueden producir.

Ejemplo:

- Crear un índice en la columna nombre de la tabla clientes

```
create index Customer_name_index on prueba_db.Customer (name);
```