Lenguaje de Manipulación

- INSERT
- DELETE
- UPDATE
- SELECT

INSERT

- Al ingresar los datos de cada registro debe tenerse en cuenta la cantidad y el orden de los campos.
- Usamos "INSERT INTO" → Especificamos los nombres de los campos entre paréntesis y separados por comas y luego, tras VALUES, los valores para cada campo, también entre paréntesis y separados por comas

```
mysql> INSERT INTO usuarios (nombre, edad) VALUES ('Fdo Torres', 30);
```

- Se pueden ingresar los valores de los campos en distinto orden del indicado en la creación de la tabla, siempre y cuando se indique **ESE orden alternativo** en el primer paréntesis.

INSERT

- Dar de alta **de forma múltiple**

```
mysql> INSERT INTO jugador (nombre, clave) VALUES
    -> ('Fdo Torres','El niño'),
    -> (‘Christiano Ronaldo’, ‘El loco’),
    -> (‘Iker Casillas’, ‘El majo’),
    -> (‘Leo Messi’, ‘El peque’)
    -> ;
```

PRÁCTICA de ejemplo

- Dar de alta varios registros en la tabla libro de la base de datos librería

```
mysql> insert into libro (titulo,autor,editorial,precio,cantidad)  values ('El niño llorón','Paco','GP',5.50,100);

mysql>  insert into libro (titulo,autor,editorial,precio,cantidad)   values ('General Mola','Capitan','Mili',25.75,50);

mysql>  insert into libro (titulo,autor,editorial,precio,cantidad)   values ('Juegos de azar'','Birgo','JPG',8.20,15);

mysql> select * from libro;
```

(Como no se dan valores para todas las columnas, el resto se rellena con NULL)

PRÁCTICA entre alumnos

- Dar de alta varios registros en alguna de las tablas de la base de datos escuela, y polideportivo

REPLACE

- Cuando intentamos ingresar con INSERT un registro que repite el valor de un campo clave, aparece un mensaje de error indicando que el valor está duplicado. Si empleamos REPLACE en lugar de INSERT, el registro existente se borra y se ingresa el nuevo, de esta manera no se duplica el valor único.

```
INSERT INTO libro (idLibro, titulo, autor, editorial, precio, cantidad)  VALUES (5, 'Fisica a diario','Aringa','tututt',15.4, 7);
```

```
INSERT INTO libro (idLibro, titulo, autor, editorial, precio, cantidad) VALUES (5,'SQL para principiantes',' Edisson','cococo',25.5, 10);
```

… dará ERROR !! En su lugar:

```
REPLACE INTO libro (idLibro, titulo, autor, editorial, precio, cantidad) VALUES (5,'SQL para ti','PepeBotella','dolar',25.5, 10);
```

DELETE

- Para eliminar los registros de una tabla.
- Borra TODOS los registros de la tabla.

```
mysql> DELETE FROM clientes;
```

- Si queremos eliminar uno o varios registros debemos indicar cuál o cuáles → WHERE condición.

```
mysql> DELETE FROM libros WHERE autor = ‘Paco’;
```

- Una vez eliminado un registro “no hay forma” de recuperarlo

UPDATE

- Modificar uno o varios valores de uno o varios registros.

```
mysql> UPDATE profesor  SET  IDprof='abcdef'
```

El cambio afectará a todos los registros

- Si queremos modificar un solo registro debemos indicar cuál o cuáles → WHERE condición.

```
mysql> UPDATE profesor SET clave='12345'  WHERE  nombre='Ricardo Menéndez';
```

- Si no encuentra registros que cumplan con la condición del "where", ningún registro es afectado.

```
mysql> UPDATE profesor SET nombre='Jorge Torralba', clave='12345''
      → WHERE  nombre='Ricardo Menéndez';
```

Modificación de varios campos

SELECT

- Para extraer información de una/varias tablas(s) de una base de datos.

SELECT [* | nomCol | nomcol1, …, nomColN] FROM nomTabla [WHERE condición(es)] [opciones]

- Para presentar los valores de una columna determinada de una tabla, usamos:
  - SELECT Nombre_de_columna FROM Nombre_de_tabla;

```
mysql>  SELECT descripcion FROM sucursales;
```

SELECT

- Para seleccionar más de una columna:
  - SELECT Nombre_de_columna_1, …, Nombre_de_columna_N FROM Nombre_de_tabla;

```
mysql>  SELECT nombre, direccion, codpostal FROM clientes;
```

- Para seleccionar todas las columnas de una tabla:
  - SELECT * FROM Nombre_de_tabla;

```
mysql>  SELECT  * FROM recursos;
```

SELECT

- Para utilizar campos existentes y campos calculados:

```
mysql>  SELECT importe, 0.10 *importe FROM ventas;
```

La expresión (0.10*importe) constituye lo que se denomina un campo calculado que se obtiene a partir de un campo/columna de la tabla.

En general es deseable no tener cabeceras de columna complicados como "0.10*Importe". Para esos casos, crearemos ALIAS, usando la instrucción AS:

```
mysql>  SELECT importe, 0.10 *importe AS Descuento FROM ventas;
```

WHERE (perteneciente a SELECT)

- Esta cláusula sirve para seleccionar filas, dentro de las columnas seleccionadas. Se pueden seleccionar filas donde una/varias columna tenga(n) un valor determinado.

```
mysql>  SELECT * FROM clientes WHERE id=1;
```

```
mysql>  SELECT * FROM ventas WHERE fecha < “2010-2-21”;
```

WHERE (perteneciente a SELECT)

Para determinar si un dato es NULL se usa la condición “IS NULL”, para saber si es no nulo se usa: “IS NOT NULL”

```
mysql>  SELECT * FROM ventas WHERE sucursal IS NOT NULL;
```

WHERE (perteneciente a SELECT)

BETWEEN

Utilizado en lugar de operadores relacionales:

```
select * from libro where precio>=20 AND precio<=40;
select * from libro where precio BETWEEN 20 and 40;
```

IN

```
select * from libro  where autor='Paco' or autor='Pepe Botella';
select * from libro  where autor IN ('Paco','Pepe Botella'');
```

NOT IN

```
select * from libro where autor NOT IN ('Paco','Pepe Botella');
```

WHERE (perteneciente a SELECT)

Operadores de comparación:

- a = b → a es igual a b
- a <> b → a es distinto de b
- a < b → a es menor que b
- a > b → a es mayor que b
- a <= b → a es menor o igual a b
- a >= b → a es mayor o igual a b
- a IS NULL → a es NULL
- a IS NOT NULL → a no es NULL

Las condiciones simples pueden aparecer combinadas por operadores lógicos. Los operadores lógicos son AND, OR (seleccionará los registros que cumplan con la primera condición, con la segunda condición o con ambas condiciones), XOR (cumplen 1 de las condiciones, no ambas) y NOT.

El operador NOT requiere paréntesis. Es decir se debe escribir WHERE NOT (salario > 50)

Trabajando con MySQL 8

Práctica entre alumnos: POR PAREJAS, hacerse querys de uno a otro (habiendo hecho una pequeña observación a los valores generales de la tabla), y ver si somos capaces de resolverlas.

Una vez, que hayamos respondido a tres/cuatro, pedir modificaciones de valores siguiendo alguna condición

USAR distintas tablas entre la pareja.

SELECT - Patrones

el operador "=" (igual), también el operador "<>" (distinto) comparan cadenas de caracteres completas.

Para comparar “porciones de cadenas” utilizamos los operadores "like" y "not like"

```
SELECT  * FROM libros  WHERE  autor LIKE "%Botella%";
```

El símbolo "%" (porcentaje) reemplaza cualquier cantidad de caracteres (incluyendo ningún caracter). Es un carácter comodín.

```
SELECT * FROM  libro  WHERE titulo NOT LIKE 'F%';
```

¿QUÉ DEVUELVE?

Así como "%" reemplaza cualquier cantidad de caracteres, el guión bajo "_" reemplaza UN caracter.

SELECT - ORDER BY

Ordenaremos el resultado de un "select" para que los registros se muestren ordenados alfabéticamente por algún campo.

```
SELECT codigo,titulo,autor,editorial,precio FROM libro ORDER BY titulo;
```

si colocamos el número de orden del campo por el que queremos que se ordene:

```
SELECT codigo,titulo,autor,editorial,precio FROM libro ORDER BY 5;
```

Por defecto, si no aclaramos en la sentencia, los ordena de manera ascendente (de menor a mayor). Podemos ordenarlos de mayor a menor, para ello agregamos la palabra clave **DESC**.

SELECT - ORDER BY

También podemos ordenar por varios campos, por ejemplo, por "título" y "editorial":

```
SELECT codigo,titulo,autor,editorial,precio FROM libros ORDER BY titulo, editorial;
```

Podemos ordenar en distintos sentido:

```
SELECT codigo,titulo,autor,editorial,precio FROM libros ORDER BY titulo asc, editorial desc;
```

Obtener de la BBDD city, todas las provincias de España, y las ciudades de cada provincia

```
SELECT district, name FROM city where countrycode='ESP' ORDER BY district;
```

SELECT - GROUP BY

Agrupa registros para consultas detalladas

```
mysql> SELECT ciudad, count(*) AS cantidad  FROM cliente GROUP BY ciudad;
```

Devolverá un listado de ciudades y el total de los clientes de cada una, agrupados por su campo ciudad

```
mysql> SELECT ciudad, count(telefono)  FROM cliente  GROUP BY ciudad;
```

Devolverá los nombres de las ciudades y el total de los clientes con esa procedencia, CON TELEFONO NO NULO, ordenados por su campo ciudad.

La función COUNT( ) retorna distintos valores cuando envíamos como argumento un asterisco o el nombre de un campo: en el primer caso cuenta todos los registros incluyendo los que tienen valor nulo, en el segundo, los registros en los cuales el campo especificado es no nulo.

SELECT - GROUP BY

Podemos agrupar por más de un campo, por ejemplo, vamos a hacerlo por "ciudad" y "sexo":

```
SELECT ciudad, sexo, count(*) FROM cliente GROUP BY  ciudad,sexo;
```

También es posible limitar la consulta con "where".

```
SELECT ciudad, count(*) FROM cliente WHERE  ciudad<>'Madrid'  GROUP BY ciudad;
```

Podemos usar las palabras claves "asc" y "desc" para una salida ordenada:

```
SELECT ciudad, count(*) FROM cliente  GROUP BY ciudad DESC;
```

SELECT - HAVING

Permite seleccionar (o rechazar) un grupo de registros ya calculados.

```
mysql> SELECT ciudad, count(*) AS cantidad  FROM cliente GROUP BY ciudad;
```

Devolverá un listado de ciudades y el total de los clientes de cada una, ordenados por su campo ciudad

```
mysql> SELECT ciudad, count(*)  FROM cliente  GROUP BY ciudad HAVING count(*)>100;
```

Devolverá un listado de ciudades y el total de los clientes de cada una, ordenados por su campo ciudad, pero SÓLO aquel grupo de ciudades que supere los cien clientes.

```
mysql> SELECT editorial, avg(precio) FROM libros  GROUP BY editorial  HAVING avg(precio)>10;
```

¿QUÉ DEVUELVE?

SELECT - HAVING

No debemos confundir la cláusula "where" con la cláusula "having";

- WHERE establece condiciones para la selección de registros de un "select";
- HAVING establece condiciones para la selección de registros de una salida "group by".
- La cláusula group by se utiliza cuando las funciones de agregación se aplican a un grupo de conjuntos de tuplas, y la cláusula having se utiliza para poner una condición a ESOS grupos.
- Si en una misma consulta aparece where y having, se aplica primero el predicado de where.

SELECT - DISTINCT

Se especifica que los registros con ciertos datos duplicados sean obviados en el resultado.

```
mysql> SELECT DISTINCT familia  FROM productos;
```

Sólo devolverá una fila por cada género existente, en la tabla

SELECT - DISTINCT EJEMPLOS

Listado de las asignaturas de los cursos sin repetición:

```
mysql> SELECT DISTINCT asignatura  FROM cursos;
```

Cursos donde la asignatura incluya "BBDD", sin repetir horario ni asignatura:

```
mysql> SELECT DISTINCT horario,asignatura FROM cursos
→ WHERE asignatura LIKE '%BBDD%';
```

Cantidad de cursos DISTINTOS agrupados por horario:

```
mysql> SELECT horario,count(DISTINCT asignatura)  FROM cursos
→ GROUP BY horario;
```

SELECT - DISTINCT **EJEMPLOS**

diferencia entre:

```
select district from city where countrycode='ESP' order by district;
```

y

```
select distinct district from city where countrycode='ESP' order by district;
```

FUNCIONES DE AGREGADO

También llamadas funciones de agrupamiento, porque operan con sobre conjuntos de registros, no con datos individuales.

Son funciones que toman una colección de valores como entrada y producen un único valor de salida.

- Funciones de agregado: AVG(), MAX(), MIN(), COUNT(), SUM()
- Funciones de agregado: UCASE(), LCASE(), MID(), CONCAT()

FUNCIONES DE AGREGADO

También llamadas funciones de agrupamiento, porque operan con sobre conjuntos de registros, no con datos individuales.

Son funciones que toman una colección de valores como entrada y producen un único valor de salida.

· Funciones de agregado más habituales:

AVG( ), MAX( ), MIN( ), COUNT( ), SUM( )

COUNT (*)

La función "count()" cuenta la cantidad de registros de una tabla, incluyendo los que tienen valor nulo, si no se especifica columna alguna como parámetro.

```
SELECT COUNT(*) FROM Cliente;
```

```
SELECT COUNT(*) FROM Cliente WHERE Ciudad_Origen= “Madrid”;
```

```
SELECT COUNT(Cantidad) FROM Productos
      WHERE nombre_Producto LIKE ‘%periferico%’;
```

Contará aquellos registros que tengan valor NO NULO en su campo “Cantidad”

SUM (*), MIN(*), MAX(*), AVG (*)

```
select sum(cantidad) from productos;
```

retorna la suma de los valores que contiene el campo especificado. SOLO datos Numéricos

```
select min(precio) from productos
        where marca='Lasika';
```

retorna el mínimo valor que contiene el campo especificado y cumple la condición impuesta

```
select max(precio) from productos;
```

retorna el máximo valor que contiene el campo especificado

```
select avg(precio) from productos
        where Ref Like '%345%';
```

retorna el valor promedio de los valores del campo especificado. SOLO datos numéricos

PRÁCTICA

Ejercicios sobre la base de datos tienda **(Ejercicio Extra 1 - Una tabla -)**

- creación de bbdd
- creación de tablas
- inclusión de registros
- realización de actualizaciones sobre la estructura de la tabla
- respuesta en consola de los comandos de consulta necesarios para satisfacer la demanda de información.

Valores por Defecto

Un valor por defecto se inserta cuando no está presente al ingresar un registro y en algunos casos en que el dato ingresado es inválido.

Para campos de cualquier tipo no declarados "not null" el valor por defecto es "null"

- Cadenas de caracteres → cadena vacía: “ ” (sin espacio entre medias).
- Valores numéricos → 0;
- En caso de ser "auto_increment" → valor mayor existente+1 (comenzando en 1).
- Campos de tipo fecha y hora, → 0 (en un campo "date" es "0000-00-00").

Valores por Defecto

Podremos establecer valores por defecto para los campos cuando creamos la tabla.
Utilizamos "DEFAULT" al definir el campo.

```
create table Productos1euro(
  codigo int unsigned auto_increment,
  nombreP varchar(40) not null,
  marca varchar(30) DEFAULT  “ ” ,
  modeloP varchar(30) DEFAULT 'Desconocido',
  cantidad int unsigned not null,
  precio decimal(4,2) unsigned DEFAULT 99.99,
  primary key (codigo)
 );
```

Para todos los tipos, excepto "blob", "text" y "auto_increment" se pueden explicitar valores por defecto con la cláusula "default”

Funciones Matemáticas

Los operadores aritméticos son "+" , "-" , "*" y "/". Todas las operaciones matemáticas retornan "null" en caso de error.

MySQL tiene algunas funciones para trabajar con números:

-ABS(x): retorna el valor absoluto del argumento "x". Ejemplo: **select abs(-20);** retorna 20.
-CEILING(x): redondea hacia arriba el argumento "x". Ejemplo: **select ceiling(12.34)**; retorna 13.
-FLOOR(x): redondea hacia abajo el argumento "x". Ejemplo: **select floor(12.34)**; retorna 12.
-GREATEST(x,y,..): retorna el argumento de máximo valor.
-LEAST(x,y,...): con dos o más argumentos, retorna el argumento más pequeño.
-MOD(n,m): significa "módulo aritmético"; retorna el resto de "n" dividido en "m". Ejemplos: **select mod(10,3);**retorna 1 . **select mod(10,2);** retorna 0.
-POWER(x,y): retorna el valor de "x" elevado a la "y" potencia. Ejemplo: **select power(2,3)**; retorna 8.

Funciones Matemáticas

-RAND(): retorna un valor de coma flotante aleatorio dentro del rango 0 a 1.0.
-ROUND(x): retorna el argumento "x" redondeado al entero más cercano. Ejemplos:select round(12.34); retorna 12. select round(12.64); retorna 13.
-SQRT(): devuelve la raíz cuadrada del valor enviado como argumento.
-TRUNCATE(x,d): retorna el número "x", truncado a "d" decimales. Si "d" es 0, el resultado no tendrá parte fraccionaria. Ejemplos:select truncate(123.4567,2);retorna 123.45;select truncate (123.4567,0);retorna 123.

Todas retornan null en caso de error.

Referencia completa de funciones:
https://dev.mysql.com/doc/refman/8.0/en/mathematical-functions.html

Funciones Matemáticas - EJEMPLOS

```
select nombreP, ceiling(precio),floor(precio)   from Producto;

select nombreP, round(precio)   from Producto;

select nombreP, truncate(precio,1)   from Producto;
```

¿Qué devuelven?

COLUMNAS CALCULADAS

Es posible obtener salidas en las cuales una columna sea el resultado de un cálculo y no un campo de una tabla.

```
SELECT nombreP,  precioP, cantidadP, precio*cantidad AS Total_Precio FROM Productos;
```

```
SELECT titulo, precio, precio*0.1, precio-(precio*0.1)
        FROM  Prodcutos;
```

¿Qué devuelve?