**UD6. SQL I. Realización de consultas sencillas.**

El **DML** (*Data Manipulation Language*) o **Lenguaje de Manipulación de Datos** es la parte de SQL dedicada a la manipulación de los datos. Las sentencias **DML** son las siguientes:

* SELECT: se utiliza para realizar consultas y extraer información de la base de datos.
* INSERT: se utiliza para insertar registros en las tablas de la base de datos.
* UPDATE: se utiliza para actualizar los registros de una tabla.
* DELETE: se utiliza para eliminar registros de una tabla.

En este tema nos vamos a centrar en el uso de la sentencia SELECT.

**Consultas básicas sobre una tabla**

Sintaxis de la instrucción **SELECT**

**SELECT** [**DISTINCT**] select\_expr [, select\_expr ...]

[**FROM** table\_references]

[**WHERE** where\_condition]

[**GROUP** **BY** {col\_name | expr | position} [**ASC** | **DESC**], ...

[**HAVING** having\_condition]

[**ORDER** **BY** {col\_name | expr | position} [**ASC** | **DESC**], ...]

[**LIMIT** {[offset,] row\_COUNT | row\_COUNT OFFSET offset}]

* Cláusula FROM.
* Cláusula WHERE (Es opcional, puede ser que no aparezca).
* Cláusula GROUP BY (Es opcional, puede ser que no aparezca).
* Cláusula HAVING (Es opcional, puede ser que no aparezca).
* Cláusula SELECT.
* Cláusula ORDER BY (Es opcional, puede ser que no aparezca).
* Cláusula LIMIT (Es opcional, puede ser que no aparezca).

Hay que tener en cuenta que el resultado de una consulta siempre será una tabla de datos. El hecho de que el resultado de una consulta sea una tabla es muy interesante porque nos permite realizar cosas como almacenar los resultados como una nueva en la base de datos. También será posible combinar el resultado de dos o más consultas para crear una tabla mayor con la unión de los dos resultados. Además, los resultados de una consulta también pueden consultados por otras nuevas consultas.

La cláusula SELECT nos permite indicar cuáles serán las columnas que tendrá la tabla de resultados de la consulta que estamos realizando. Las opciones que podemos indicar son las siguientes.

Supongamos que tenemos una tabla llamada **alumno** con la siguiente información de los alumnos matriculados en un determinado curso.



Veamos qué consultas serían necesario realizar para obtener los siguientes datos.

**1. Obtener todos los datos de todos los alumnos matriculados en el curso.**

**SELECT** \*

**FROM** alumno;

El carácter \* es un comodín que utilizamos para indicar que queremos seleccionar todas las columnas de la tabla. La consulta anterior devolverá todos los datos de la tabla.

Tenga en cuenta que las palabras reservadas de SQL no son case sensitive, por lo tanto es posible escribir la sentencia en mayúsculas o minúsculas y el resultado será el mismo.

**2. Obtener el nombre de todos los alumnos.**

**SELECT** nombre

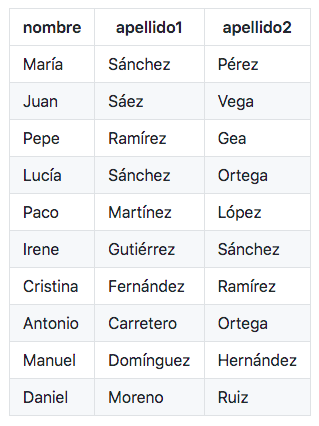
**FROM** alumno;



**3. Obtener el nombre y los apellidos de todos los alumnos.**

**SELECT** nombre, apellido1, apellido2

**FROM** alumno;



* **Cómo obtener columnas calculadas**

Es posible realizar cálculos aritméticos entre columnas para calcular nuevos valores. Por ejemplo, supongamos que tenemos la siguiente tabla con información sobre ventas.



Y queremos calcular una nueva columna con el precio total de la venta, que sería equivalente a multiplicar el valor de la columna cantidad\_comprada por precio\_por\_elemento.

Para obtener esta nueva columna podríamos realizar la siguiente consulta:

**SELECT** **id**, cantidad\_comprada, precio\_por\_elemento, cantidad\_comprada \* precio\_por\_elemento

**FROM** ventas;

Y el resultado sería el siguiente:



* **Cómo realizar alias de columnas con AS**

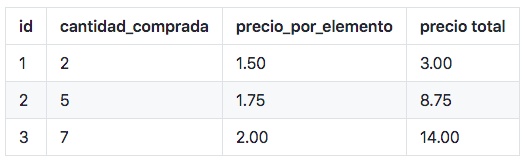
Con la palabra reservada AS podemos crear alias para las columnas. Esto puede ser útil cuando estamos calculando nuevas columnas a partir de valores de las columnas actuales. En el ejemplo anterior de la tabla que contiene información sobre las ventas, podríamos crear el siguiente alias:

**SELECT** **id**, cantidad\_comprada, precio\_por\_elemento, cantidad\_comprada \* precio\_por\_elemento **AS** 'precio total'

**FROM** ventas;

Si el nuevo nombre que estamos creando para el alias contiene espacios en blanco es necesario usar comillas simples.

Al crear este alias obtendremos el siguiente resultado:



* **Modificadores ALL y DISTINCT**

Los modificadores ALL y DISTINCT indican si se deben incluir o no filas repetidas en el resultado de la consulta.

* ALL indica que se deben incluir todas las filas, incluidas las repetidas. Es la

opción por defecto, por lo tanto no es necesario indicarla.

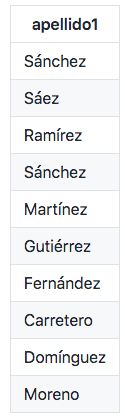
* DISTINCT elimina las filas repetidas en el resultado de la consulta.

**Ejemplo**

En el siguiente ejemplo vamos a ver la diferencia que existe entre utilizar DISTINCT y no utilizarlo. La siguiente consulta mostrará todas las filas que existen en la columna apellido1 de la tabla alumno.

**SELECT** apellido1

**FROM** alumno;



Si en la consulta anterior utilizamos DISTINCT se eliminarán todos los valores repetidos que existan.

**SELECT** **DISTINCT** apellido1

**FROM** alumno;



* **Cláusula ORDER BY**

ORDER BY permite ordenar las filas que se incluyen en el resultado de la consulta. La sintaxis de MySQL es la siguiente:

[**ORDER** **BY** {col\_name | expr | position} [**ASC** | **DESC**], ...]

Esta cláusula nos permite ordenar el resultado de forma ascendente ASC o descendente DESC, además de permitirnos ordenar por varias columnas estableciendo diferentes niveles de ordenación.

### Cómo ordenar de forma ascendente

**Ejemplo**

**Obtener el nombre y los apellidos de todos los alumnos, ordenados por su nombre de forma ascendente.**

**SELECT** apellido1, apellido2, nombre

**FROM** alumno

**ORDER** **BY** nombre;

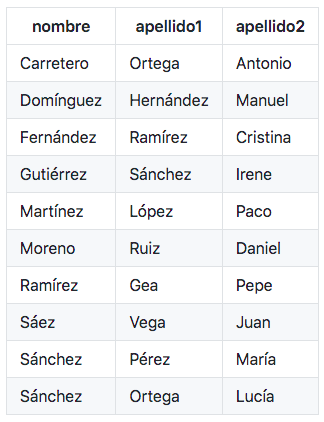
* Si no indicamos nada en la cláusula ORDER BY se ordenará por defecto de forma ascendente.
* La consulta anterior es equivalente a esta otra.

**SELECT** apellido1, apellido2, nombre

**FROM** alumno

**ORDER** **BY** nombre **ASC**;

El resultado de ambas consultas será:



### Cómo ordenar de forma descendente

**Ejemplo**

**Obtener el nombre y los apellidos de todos los alumnos, ordenados por su primer apellido de forma descendente.**

**SELECT** apellido1, apellido2, nombre

**FROM** alumno

**ORDER** **BY** apellido1 **DESC**;

* **Cláusula LIMIT**

LIMIT permite limitar el número de filas que se incluyen en el resultado de la consulta. La sintaxis de MySQL es la siguiente

[**LIMIT** {[offset,] row\_COUNT | row\_COUNT OFFSET offset}]

donde row\_COUNT es el número de filas que queremos obtener y offset el número de filas que nos saltamos antes de empezar a contar. Es decir, la primera fila que se obtiene como resultado es la que está situada en la posición offset + 1.

* **Cláusula WHERE**

La cláusula WHERE nos permite añadir filtros a nuestras consultas para seleccionar sólo aquellas filas que cumplen una determinada condición. Estas condiciones se denominan predicados y el resultado de estas condiciones puede ser **verdadero**, **falso** o **desconocido**.

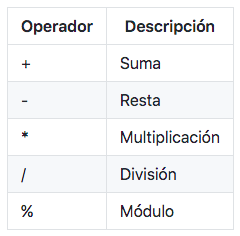
Una condición tendrá un resultado **desconocido** cuando alguno de los valores utilizados tiene el valor NULL.

Los operandos usados en las condiciones pueden ser nombres de columnas, constantes o expresiones. Los operadores que podemos usar en las condiciones pueden ser aritméticos, de comparación, lógicos, etc.

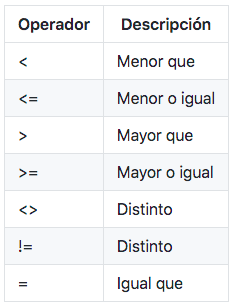
**Operadores disponibles en MySQL**

A continuación se muestran los operadores más utilizados en MySQL para realizar las consultas.

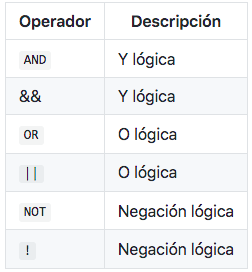
**Operadores aritméticos**



**Operadores de comparación**



**Operadores lógicos**

****

**Ejemplos**

Vamos a continuar con el ejemplo de la tabla **alumno** que almacena la información de los alumnos matriculados en un determinado curso.

¿Qué consultas serían necesarias para obtener los siguientes datos?

**1. Obtener el nombre de todos los alumnos que su primer apellido sea Martínez**

**SELECT** nombre

**FROM** alumno

**WHERE** apellido1 = 'Martínez';

**2. Obtener todos los datos del alumno que tiene un id igual a 9.**

**SELECT** \*

**FROM** alumno

**WHERE** **id** = 9;

**3. Obtener el nombre y la fecha de nacimiento de todos los alumnos que nacieron después del 1 de enero de 1997.**

**SELECT** nombre, fecha\_nacimiento

**FROM** alumno

**WHERE** fecha\_nacimiento >= '1997/01/01';

**Ejercicios propuestos**

Realice las siguientes consultas teniendo en cuenta la **base de datos instituto** (tabla alumno)

1. Devuelve los datos del alumno cuyo id es igual a 1.
2. Devuelve los datos del alumno cuyo teléfono es igual a 692735409.
3. Devuelve un listado de todos los alumnos que son repetidores.
4. Devuelve un listado de todos los alumnos que no son repetidores.
5. Devuelve el listado de los alumnos que han nacido antes del 1 de enero de 1993.
6. Devuelve el listado de los alumnos que han nacido después del 1 de enero de 1994.
7. Devuelve el listado de los alumnos que han nacido después del 1 de enero de 1994 y no son repetidores.

**Operador BETWEEN**

**Sintaxis:**

expresión [**NOT**] **BETWEEN** cota\_inferior **AND** cota\_superior

Se utiliza para comprobar si un valor está dentro de un rango de valores. Por ejemplo, si queremos obtener los pedidos que se han realizado durante el mes de enero de 2018 podemos realizar la siguiente consulta:

**SELECT** \*

**FROM** pedido

**WHERE** fecha\_pedido **BETWEEN** '2018-01-01' **AND** '2018-01-31';

**Ejercicios propuestos**

Realice las siguientes consultas teniendo en cuenta la **base de datos instituto**.

1. Devuelve los datos de los alumnos que hayan nacido entre el 1 de enero de 1998 y el 31 de mayo de 1998.
2. Devuelve los datos de los alumnos que **no** hayan nacido entre el 1 de enero de 1998 y el 31 de mayo de 1998.

**Operador IN**

Este operador nos permite comprobar si el valor de una determinada columna está incluido en una lista de valores.

**Ejemplo:**

Obtener todos los datos de los alumnos que tengan como primer apellido Sánchez, Martínez o Domínguez.

**SELECT** \*

**FROM** alumno

**WHERE** apellido1 **IN** (`Sánchez`, `Martínez`, `Domínguez`);

**Ejemplo:**

Obtener todos los datos de los alumnos que **no tengan** como primer apellido Sánchez, Martínez o Domínguez.

**SELECT** \*

**FROM** alumno

**WHERE** apellido1 **NOT** **IN** (`Sánchez`, `Martínez`, `Domínguez`);

**Operador LIKE**

Sintaxis:

columna [**NOT**] **LIKE** patrón

Se utiliza para comparar si una cadena de caracteres coincide con un patrón. En el patrón podemos utilizar cualquier carácter alfanumérico, pero hay dos caracteres que tienen un significado especial, el símbolo del porcentaje (%) y el carácter de subrayado (\_).

* %: Este carácter equivale a cualquier conjunto de caracteres.
* \_: Este carácter equivale a cualquier carácter.

**Ejemplos:**

Devuelva un listado de todos los alumnos que su primer apellido empiece por la letra S.

**SELECT** \*

**FROM** alumno

**WHERE** apellido1 **LIKE** 'S%';

Devuelva un listado de todos los alumnos que su primer apellido termine por la letra z.

**SELECT** \*

**FROM** alumno

**WHERE** apellido1 **LIKE** '%z';

Devuelva un listado de todos los alumnos que su nombre tenga el carácter a.

**SELECT** \*

**FROM** alumno

**WHERE** nombre **LIKE** '%a%';

Devuelve un listado de todos los alumnos cuyo nombre empieza con estas cuatro letras ‘A%BC’.

En este caso, el patrón que queremos buscar contiene el carácter %, por lo tanto, tenemos que usar un carácter de escape.

**SELECT** \*

**FROM** producto

**WHERE** nombre **LIKE** 'A$%BC%' **ESCAPE** '$';

**Operador IS e IS NOT**

Estos operadores nos permiten comprobar si el valor de una determinada columna es NULL o no lo es.

**Ejemplo:**

Obtener la lista de alumnos que tienen un valor NULL en la columna teléfono.

**SELECT** \*

**FROM** alumno

**WHERE** teléfono **IS** **NULL**;

**Ejemplo:**

Obtener la lista de alumnos que no tienen un valor NULL en la columna teléfono.

**SELECT** \*

**FROM** alumno

**WHERE** teléfono **IS** **NOT** **NULL**;