T05. MySQL - Consultas sobre una tabla.

DML (Data Manipulation Language) es el Lenguaje de Manipulación de datos de SQL. Como indica su nombre está compuesto por las instrucciones o sentencias que permiten la manipulación de los datos y son las siguientes:

- SELECT. Permite realizar consultas y extraer información de la base de datos.
- INSERT. Permite insertar registros en una tabla.
- UPDATE. Permite actualizar registros de una tabla.
- DELETE. Permite borrar registros de una tabla.

Por tanto, para realizar las consultas utilizaremos la sentencia SELECT.

Algunas observaciones:

En determinadas circunstancias puede que limpiar la pantalla nos facilite el trabajo, para ello podemos usar: "**Ctrol + L**".

Inclusión de comentarios. Los comentarios son útiles para documentar el funcionamiento de los comandos, en particular cuando se realizan scripts. En MySQL hay varias formas de especificar un comentario:

- -- Doble guión al principio de una línea y toda la línea se considera comentario.
- # Se considera comentario todo lo que se escriba a continuación. Puede aparecer en cualquier lugar, normalmente detrás de una sentencia.
- /* .. */ A partir de /* se considera comentario hasta que se encuentre */, este tipo de comentario puede ocupar más de una línea.

5.1. Sintaxis de SELECT.

La sintaxis completa incluye elementos que pueden dificultar el aprendizaje inicial, por ello veamos una sintaxis simplificada que se irá completando. Siempre se puede consultar la documentación oficial de MySQL, allí sí estará todo y actualizado.

Sintaxis simplificada SELECT:

```
SELECT [DISTINCT] expresion_select [, expresion_select ..]
    [FROM nombre_tabla]
    [WHERE condicion_where]
    [GROUP BY {nombre_col | expresión | posición} [ASC | DESC],.. [WITH ROLLUP]]
    [HAVING condicion_where]
    [ORDER BY {nombre_col | expresión | posición} [ASC | DESC],..]
    [LIMIT {[offset,] row_COUNT | row_COUNT OFFSET offset}]
```

Al realizar una consulta el resultado siempre será una tabla, que podrá tener una, varias o ninguna columna y una, varias o ninguna fila.

Esta situación ofrece cierta versatilidad ya que al tener como resultado una tabla esta se puede almacenar en otra base de datos, se pueden combinar el resultado de varias consultas, incluso los resultados de una consulta pueden ser la entrada para otra nueva consulta.

expresion_select:

Con expresion_select se va a indicar todas las columnas que tendrá la tabla resultado de la consulta y puede ser:

- Un nombre de la columna de la tabla a consultar. Esa tabla se especifica detrás de la cláusula FROM.
- Una constante, ese valor constante aparecerá en todas las filas.
- Una expresión, está nos permitirá realizar cálculos que se mostrarán en la consulta, cada uno en su fila correspondiente.
- *, mostrará todas las columnas de la tabla a consultar. Si se utiliza esta opción no se pueden poner ninguna otra seleccion_select. Directamente detrás aparecerá: FROM nombre_tabla.

Para realizar algunas pruebas vamos a partir de la siguiente tabla:

```
DROP DATABASE IF EXISTS institutodb;
CREATE DATABASE institutodb CHARACTER SET utf8mb4;
USE institutodb;

CREATE TABLE alumno
(
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  apellidol VARCHAR(100) NOT NULL,
  apellido2 VARCHAR(100),
```

```
 fecha_nacimiento DATE NOT NULL,
 es_repetidor ENUM('si', 'no') NOT NULL,
 telefono VARCHAR(9)
);

INSERT INTO alumno VALUES(1, 'Maria', 'Sanchez', 'Perez', '1990/12/01', 'no', NULL);
INSERT INTO alumno VALUES(2, 'Juan', 'Saezt', 'Vega', '1998/04/02', 'no', 618253876);
INSERT INTO alumno VALUES(3, 'Pepe', 'Ramirez', 'Gea', '1988/01/03', 'no', NULL);
INSERT INTO alumno VALUES(4, 'Lucia', 'Sanchez', 'Ortega', '1993/06/13', 'si', 678516294);
INSERT INTO alumno VALUES(5, 'Paco', 'Martinez', 'Lopez', '1995/11/24', 'no', 692735409);
INSERT INTO alumno VALUES(6, 'Irene', 'Gutierrez', 'Sanchez', '1991/03/28', 'si', NULL);
INSERT INTO alumno VALUES(7, 'Cristina', 'Fernandez', 'Ramirez', '1996/09/17', 'no', 628349590);
INSERT INTO alumno VALUES(8, 'Antonio', 'Carretero', 'Ortega', '1994/05/20', 'si', 612345633);
INSERT INTO alumno VALUES(9, 'Manuel', 'Dominguez', 'Hernandez', '1999/07/08', 'no', NULL);
INSERT INTO alumno VALUES(10, 'Daniel', 'Moreno', 'Ruiz', '1998/02/03', 'no', NULL);
```

Para consultar el contenido de la tabla completa de los alumnos:

```
mysql> use institutodb;
Database changed
mysql> select * from alumno;
+----+----------+-----------+-----------+-----------------+--------------+-----------+
| id | nombre   | apellidol | apellido2 | fecha_nacimiento | es_repetidor | telefono  |
+----+----------+-----------+-----------+-----------------+--------------+-----------+
|  1 | Maria    | Sanchez   | Perez     | 1990-12-01       | no           | NULL      |
|  2 | Juan     | Saez      | Vega      | 1998-04-02       | no           | 618253876 |
|  3 | Pepe     | Ramirez   | Gea       | 1988-01-03       | no           | NULL      |
|  4 | Lucia    | Sanchez   | Ortega    | 1993-06-13       | si           | 678516294 |
|  5 | Paco     | Martinez  | Lopez     | 1995-11-24       | no           | 692735409 |
|  6 | Irene    | Gutierrez | Sanchez   | 1991-03-28       | si           | NULL      |
|  7 | Cristina | Fernandez | Ramirez   | 1996-09-17       | no           | 628349590 |
|  8 | Antonio  | Carretero | Ortega    | 1994-05-20       | si           | 612345633 |
|  9 | Manuel   | Dominguez | Hernandez | 1999-07-08       | no           | NULL      |
| 10 | Daniel   | Moreno    | Ruiz      | 1998-02-03       | no           | NULL      |
+----+----------+-----------+-----------+-----------------+--------------+-----------+
10 rows in set (0.00 sec)
```

Como se puede ver para consultar todas las columnas se ha utilizado el comodín *

```
mysql> SELECT * FROM alumno;
mysql> select * from alumno;
```

Nótese que MySQL es “case insensitive” por lo que se pueden poner los comandos tanto en mayúsculas como en minúsculas, no las palabras definidas por el usuario. El nombre de la tabla es “alumno” en minúsculas, y así se ha de poner.

Por otra parte, un comando termina cuando aparece el “;”, esto permite que se pueda escribir en varias líneas y así es aconsejable para facilitar la compresión y lectura de los mismos, máxime cuando sean largos y se utilicen varias cláusulas. Una buena estrategia es poner cada cláusula en una línea.

5.1.1. Mostrar varias columnas.

Realicemos una consulta de los alumnos donde solo aparezcan: los apellidos y el nombre:

```
mysql>SELECT apellido1, apellido2, nombre FROM alumno;
```

```
mysql>
mysql> use institutodb;
Database changed
mysql> SELECT apellido1, apellido2, nombre FROM alumno;
+-----------+-----------+----------+
| apellido1 | apellido2 | nombre   |
+-----------+-----------+----------+
| Sanchez   | Perez     | Maria    |
| Saez      | Vega      | Juan     |
| Ramirez   | Gea       | Pepe     |
| Sanchez   | Ortega    | Lucia    |
| Martinez  | Lopez     | Paco     |
| Gutierrez | Sanchez   | Irene    |
| Fernandez | Ramirez   | Cristina |
| Carretero | Ortega    | Antonio  |
| Dominguez | Hernandez | Manuel   |
| Moreno    | Ruiz      | Daniel   |
+-----------+-----------+----------+
10 rows in set (0.00 sec)
```

El resultado mostrará los campos en el orden en el que aparecen en SELECT, independientemente de como se hayan definido en la tabla.

5.1.2. Columnas calculadas.

Puede ser que se requiera mostrar en una columna el resultado de operar con otras.

Ejemplo con una operación carácter. Para concatenar dos valores carácter se utiliza la función CONCAT(). Esta genera una nueva cadena de caracteres a partir de las que se le pasen como parámetros. Mostremos el identificador de los alumnos, sus apellidos y sus nombres:

mysql>SELECT id, CONCAT(apellido1, " ", apellido2), nombre FROM alumno;

```
mysql>
mysql> SELECT id, CONCAT(apellido1, " ",apellido2), nombre FROM alumno;
+----+-----------------------------------+----------+
| id | CONCAT(apellido1, " ",apellido2) | nombre   |
+----+-----------------------------------+----------+
|  1 | Sanchez Perez                     | Maria    |
|  2 | Saez Vega                         | Juan     |
|  3 | Ramirez Gea                       | Pepe     |
|  4 | Sanchez Ortega                    | Lucia    |
|  5 | Martinez Lopez                    | Paco     |
|  6 | Gutierrez Sanchez                 | Irene    |
|  7 | Fernandez Ramirez                 | Cristina |
|  8 | Carretero Ortega                  | Antonio  |
|  9 | Dominguez Hernandez               | Manuel   |
| 10 | Moreno Ruiz                       | Daniel   |
+----+-----------------------------------+----------+
10 rows in set (0.00 sec)
```

Ejemplo con una operación numérica: supongamos que tenemos una tabla de **ventas** que tiene una columna con la cantidad de artículos vendidos y otra con el precio unitario de ese artículo: **cantidad** y **precio_unitario** respectivamente. Se puede mostrar una tercera columna con el total:

mysql> SELECT cantidad, precio_unitario, cantidad*precio_unitario from ventas;

5.1.3. Mejorar las cabeceras: Alias de columna AS.

En las cabeceras de las columnas calculadas se muestra el cálculo que se realiza, esto se puede mejorar si indica el nombre de lo que se muestra, y no como se ha obtenido.

En el caso del listado de alumnos: Apellidos, en lugar de CONCAT(apellido1, " ", apellido2)
En el caso del las ventas: Total, en lugar de cantidad * precio_unitario.

En ese caso tanto Total como Apellidos sería alias de la operación que representan.

La forma de hacerlo es poner después de la expresión u operación AS y el texto sustituto o alias.

mysql>SELECT id, CONCAT(apellido1, " ", apellido2) AS Apellidos, nombre FROM alumno.

mysql>SELECT cantidad, precio_unitario, cantidad*precio_unitario AS Total from ventas;

```
mysql>
mysql> USE institutodb;
Database changed
mysql> SELECT id, CONCAT(apellido1, " ", apellido2) AS Apellidos, nombre FROM alumno;
+----+--------------------+----------+
| id | Apellidos          | nombre   |
+----+--------------------+----------+
|  1 | Sanchez Perez      | Maria    |
|  2 | Saez Vega          | Juan     |
|  3 | Ramirez Gea        | Pepe     |
|  4 | Sanchez Ortega     | Lucia    |
|  5 | Martinez Lopez     | Paco     |
|  6 | Gutierrez Sanchez  | Irene    |
|  7 | Fernandez Ramirez  | Cristina |
|  8 | Carretero Ortega   | Antonio  |
|  9 | Dominguez Hernandez | Manuel   |
| 10 | Moreno Ruiz        | Daniel   |
+----+--------------------+----------+
10 rows in set (0.00 sec)
```

Si el alias, es decir, el texto que aparecerá como cabecera de columna incluye espacios en blanco, irá entre comillas. Ejemplo: precio_unitario, cantidad*precio_unitario AS “Total del artículo”.

5.1.4 Modificadores ALL, DISTINCT y DISTINCTROW.

Cuando se hace una consulta puede ocurrir que aparezcan filas repetidas. No significa que tengamos registros repetidos, cosa que no debe ocurrir, sino que hay más de un registro que contienen los mismos valores en los campos consultados.

Ej: de una tabla de alumnos que consultemos solo el primer apellido, si hay varios Sánchez, saldrá una línea por cada uno de ellos.

En esta situación se puede especificar que se muestre o no las filas repetidas:

- ALL: Indicará que se muestren todas las filas. Es la opción por defecto.
- DISTINCT: Evitará repetir filas. Se especifica justo detrás de SELECT.
- DISTINCTROW: Es sinónimo de DISTINCT.

```
mysql> SELECT apellido1 FROM alumno;
+-----------+
| apellido1 |
+-----------+
| Sanchez   |
| Saez      |
| Ramirez   |
| Sanchez   |
| Martinez  |
| Gutierrez |
| Fernandez |
| Carretero |
| Dominguez |
| Moreno    |
+-----------+
10 rows in set (0.00 sec)
```

```
mysql> SELECT DISTINCT apellido1 FROM alumno;
+-----------+
| apellido1 |
+-----------+
| Sanchez   |
| Saez      |
| Ramirez   |
| Martinez  |
| Gutierrez |
| Fernandez |
| Carretero |
| Dominguez |
| Moreno    |
+-----------+
9 rows in set (0.00 sec)
```

5.1.5. Cláusula ORDER BY.

Los resultados de una consulta se pueden mostrar ordenados ascendente o descendentemente por alguna columna o expresión. Incluso se pueden establecer más de un criterio de ordenación. Estos criterios se separan por "," y se atienden por orden de aparición.

La cláusula ORDER BY nos permite esta posibilidad.

Sintaxis:

[ORDER BY {nombre_col | expresión | posición} [ASC | DESC],..]

Ejemplo: mostrar ordenado alfabéticamente la lista de alumnos de institutodb: primer apellido, segundo apellido y nombre.

```
mysql> USE institutodb;
Database changed
mysql> SELECT * FROM alumno ORDER BY apellido1, apellido2, nombre;
+----+----------+-----------+-----------+-----------------+--------------+-----------+
| id | nombre   | apellido1 | apellido2 | fecha_nacimiento | es_repetidor | telefono  |
+----+----------+-----------+-----------+-----------------+--------------+-----------+
|  8 | Antonio  | Carretero | Ortega    | 1994-05-20       | si           | 612345633 |
|  9 | Manuel   | Dominguez | Hernandez | 1999-07-08       | no           | NULL      |
|  7 | Cristina | Fernandez | Ramirez   | 1996-09-17       | no           | 628349590 |
|  6 | Irene    | Gutierrez | Sanchez   | 1991-03-28       | si           | NULL      |
|  5 | Paco     | Martinez  | Lopez     | 1995-11-24       | no           | 692735409 |
| 10 | Daniel   | Moreno    | Ruiz      | 1998-02-03       | no           | NULL      |
|  3 | Pepe     | Ramirez   | Gea       | 1988-01-03       | no           | NULL      |
|  2 | Juan     | Saez      | Vega      | 1998-04-02       | no           | 618253876 |
|  1 | Maria    | Sanchez   | NULL      | 1990-12-01       | no           | NULL      |
|  4 | Lucia    | Sanchez   | Ortega    | 1993-06-13       | si           | 678516294 |
+----+----------+-----------+-----------+-----------------+--------------+-----------+
10 rows in set (0.00 sec)
```

Por defecto se toma la opción ASC.

Observa que los campos NULL en orden ascendente van los primeros. En el ejemplo, cuando el primer apellido se repite: Sánchez se ordena ascendentemente por el segundo, apareciendo primero NULL y después Ortega.

En lugar de indicar el nombre de las columnas en la cláusula ORDER BY se puede especificar la posición en la que aparece en la orden SELECT la misma consulta anterior quedaría:

```
mysql> SELECT * FROM alumno ORDER BY 3,4,2;
```

5.1.6. Cláusula LIMIT.

Esta cláusula nos permite controlar el número de filas o líneas que se mostrarán al ejecutar la orden SELECT. El motivo de establecer esta limitación puede ser simplemente que el resultado completo no cabe en fracción de pantalla que tenemos disponible. Solo se deben mostrar las que quepan.

La sintaxis:

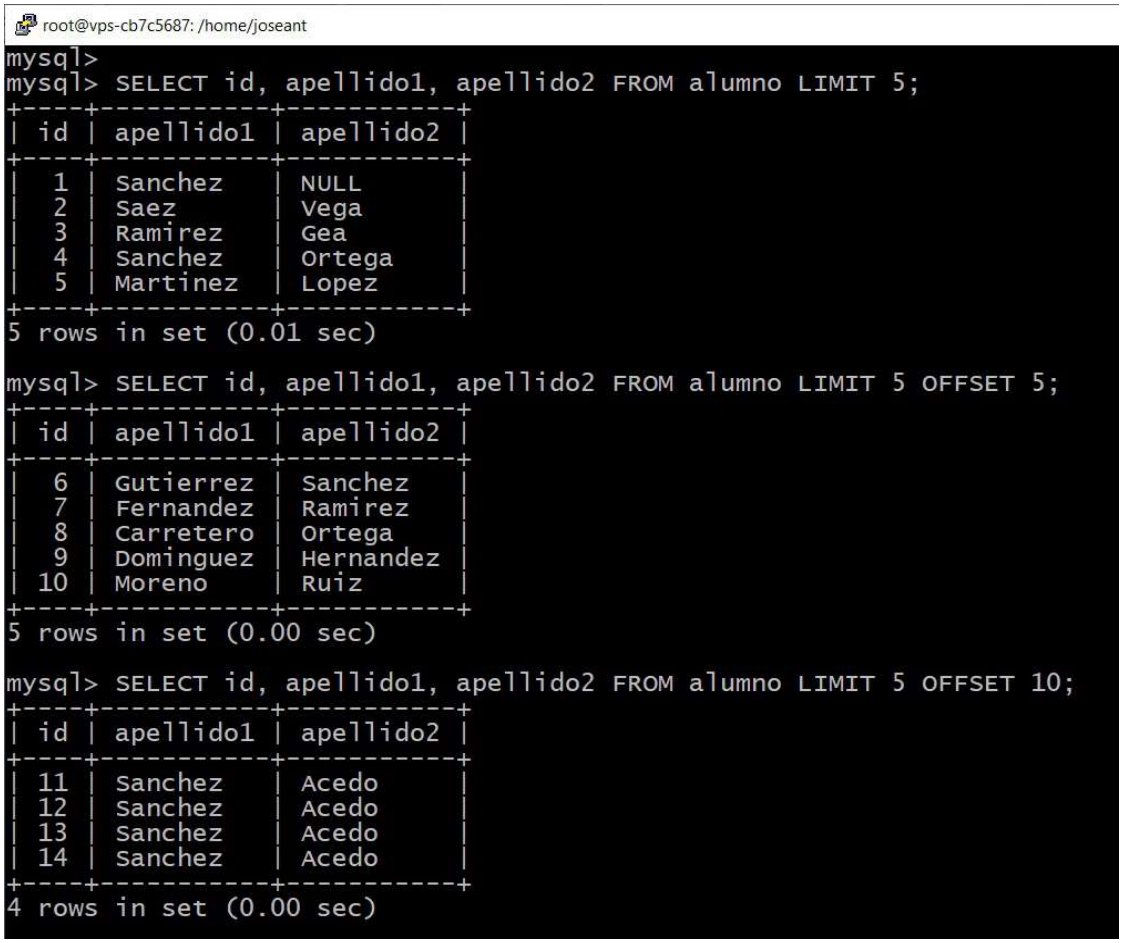
[LIMIT {[offset,] **row_COUNT | row_COUNT** OFFSET **offset**}]

row_COUNT: Indica el número de filas a mostrar.
Offset: Es el número de filas que se salta y por tanto no las mostrará, incluida ella misma. Es decir, la primera fila que se muestra es la número: offset+1.

Ejemplo: Mostrar un listado de los alumnos de nuestra base de datos de 5 en 5:

```
mysql> SELECT id, apellido1, apellido2 FROM alumno LIMIT 5;
mysql> SELECT id, apellido1, apellido2 FROM alumno LIMIT 5 OFFSET 5;
```

mysql> SELECT id, apellido1, apellido2 FROM alumno LIMIT 5 OFFSET 10;



5.1.7. Cláusula WHERE.

Mediante la cláusula WHERE se establecen filtros en las consultas para seleccionar sólo aquellas filas que cumplan una determinada condición. El resultado de una condición puede ser verdadero, falso o desconocido. Se obtendrá un resultado desconocido cuando algún valor utilizado en la condición sea NULL.

La condición se aplica a cada fila y se muestra solo aquellas que tengan un resultado verdadero.

Se pueden diferenciar cinco tipos de condiciones disponibles en la cláusula WHERE:

1. Condiciones para comparar valores o expresiones.
2. Condiciones para comprobar si un valor está dentro de un rango de valores.
3. Condiciones para comprobar si un valor está dentro de un conjunto de valores.
4. Condiciones para comparar cadenas con patrones.
5. Condiciones para comprobar si una columna tiene valores a NULL.

Operadores aritméticos, de comparación y lógicos.

Nos permiten construir las expresiones que formarán parte de las condiciones de WHERE. Con ellos podremos realizar cálculos y comparaciones. En una expresión se pueden poner tantos

operadores como sea necesario, pero debemos de tener en cuenta el orden en el que se evaluarán. Dependiendo del orden de evaluación se pueden obtener resultados distintos. Para controlar el orden de evaluación se utilizan paréntesis ( ). Evaluándose primero las expresiones situadas en los paréntesis más internos.

Operadores ARITMÉTICOS:

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo. Resto de una división. Numerador y denominador han de ser números enteros.
mod	Igual que %. Resto de una división. Se ha de dejar un espacio en blanco entre "mod" y los números.

Operadores de COMPARACIÓN.

Operador	Descripción
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
<>	Distinto
!=	Distinto
=	Igual que

Operadores LÓGICOS.

Operador	Descripción
AND	Y lógica
&&	Y lógica
OR	O lógica
||	O lógica
NOT	Negación lógica
!	Negación lógica

Los operadores lógicos están basados en el álgebra de Boole (lógica matemática). Su funcionamiento es el siguiente:

Disponemos de dos valores de verdad: FALSO ó 0 y VERDADERO ó 1

Operaciones básicas: not, and y or.

Las tablas de las operaciones NOT, AND y OR son:

NOT		**AND**			**OR**		
Ent	**Resultado**	Ent 1	Ent 2	**Resultado**	Ent 1	Ent 2	**Resultado**
0	**1**	0	0	**0**	0	0	**0**
1	**0**	0	1	**0**	0	1	**1**
		1	0	**0**	1	0	**1**
		1	1	**1**	1	1	**1**

Ejemplos:

Tengo una bicicleta	NOT (Tengo una bicicleta)
0	1
1	0

Para ir al trabajo. (Tengo vehículo propio) OR (Alguien me lleva)

0	0	0
0	1	1
1	1	0
1	1	1

Para practicar un deporte: (Tengo tiempo) AND (Quiero hacer deporte)

0	0	0
0	0	1
1	0	0
1	1	1

Ejemplos:

Listado de alumnos mayores de 27 años. Suponemos que hoy es 22 de enero de 2022.

```
mysql> SELECT id, nombre, apellido1, apellido2, fecha_nacimiento as "Fecha Nac." FROM alumno
    ->        WHERE fecha_nacimiento < "1995-01-22";
+----+---------+-----------+-----------+------------+
| id | nombre  | apellido1 | apellido2 | Fecha Nac. |
+----+---------+-----------+-----------+------------+
|  1 | Maria   | Sanchez   | NULL      | 1990-12-01 |
|  3 | Pepe    | Ramirez   | Gea       | 1988-01-03 |
|  4 | Lucia   | Sanchez   | Ortega    | 1993-06-13 |
|  6 | Irene   | Gutierrez | Sanchez   | 1991-03-28 |
|  8 | Antonio | Carretero | Ortega    | 1994-05-20 |
+----+---------+-----------+-----------+------------+
5 rows in set (0.00 sec)
```

Listado de los alumnos repetidores de los primeros 10 de la lista.

```
mysql> SELECT id, nombre, apellido1, apellido2, es_repetidor as Repite FROM alumno
    ->        WHERE id<=10 AND es_repetidor="si";
+----+---------+-----------+-----------+--------+
| id | nombre  | apellido1 | apellido2 | Repite |
+----+---------+-----------+-----------+--------+
|  4 | Lucia   | Sanchez   | Ortega    | si     |
|  6 | Irene   | Gutierrez | Sanchez   | si     |
|  8 | Antonio | Carretero | Ortega    | si     |
+----+---------+-----------+-----------+--------+
3 rows in set (0.00 sec)
```

Operador Between.

Se utiliza para comprobar si un valor está dentro de un rango de valores. En el resultado también se incluyen si coinciden con los extremos del rango.

Ejemplo para mostrar todos los alumnos nacidos entre 1995 y el 2000:

```
mysql> SELECT id, nombre, apellido1, fecha_nacimiento FROM alumno
            WHERE fecha_nacimiento BETWEEN '1995-01-01' AND '1999-12-31';
```

Que daría el mismo resultado con:

```
mysql> SELECT id, nombre, apellido1, fecha_nacimiento FROM alumno
            WHERE fecha_nacimiento > '1995-01-01' AND fecha_nacimiento < '2000-01-01';
```

```
mysql> SELECT id, nombre, apellido1, fecha_nacimiento FROM alumno
    ->    WHERE fecha_nacimiento BETWEEN '1995-01-10' AND '1999-12-31';
+----+----------+-----------+------------------+
| id | nombre   | apellido1 | fecha_nacimiento |
+----+----------+-----------+------------------+
|  2 | Juan     | Saez      | 1998-04-02       |
|  5 | Paco     | Martinez  | 1995-11-24       |
|  7 | Cristina | Fernandez | 1996-09-17       |
|  9 | Manuel   | Dominguez | 1999-07-08       |
| 10 | Daniel   | Moreno    | 1998-02-03       |
| 11 | Pepe     | Sanchez   | 1997-05-23       |
| 12 | Samuel   | Sanchez   | 1998-11-23       |
+----+----------+-----------+------------------+
7 rows in set (0.01 sec)
```

Operador IN.

Permite comprobar si el valor de una determinada columna está en una lista de valores. La lista de valores se recoge entre paréntesis.

Ej: listado de todos los alumnos cuyo primer apellido sea: García, Sánchez o Saez.

```
mysql> SELECT id, nombre, apellido1 FROM alumno
            WHERE apellido1 IN ('Garcia', 'Sanchez', 'Saez');
```

Tendría su equivalente con el operador lógico OR:

```
mysql> SELECT id, nombre, apellido1 FROM alumno
            WHERE apellido1='Garcia' OR apellido1='Sanchez' OR apellido1='Saez';
```

Se puede aplicar la negación al operador IN, en cuyo caso se filtrarían cuando no sea ninguno de los que aparecen en la lista.

Ej: listado de todos los alumnos cuyo primer apellido no sea: ni García, ni Sánchez ni Saez.

```
mysql> SELECT id, nombre, apellido1 FROM alumno
            WHERE apellido1 NOT IN ('Garcia', 'Sanchez', 'Saez');
```

Operador LIKE.

Se utiliza para comparar si una cadena coincide con un patrón. En el patrón podemos utilizar caracteres alfanuméricos, pero hay dos con un significado especial:

- % Equivale a cualquier conjunto de caracteres.
- _ Equivale a cualquier carácter.

Ejemplo: Consulta los alumnos que empiecen por la letra 'S':

```
mysql> SELECT id, nombre, apellido1 FROM alumno
            WHERE apellido1 LIKE 'S%';
```

Listado de alumnos cuyo segundo apellido tenga 4 letras:

```
mysql> SELECT id, nombre, apellido1 FROM alumno
            WHERE apellido1 LIKE '_ _ _ _';
```

En caso de tener que utilizar en la cadena el carácter “%” o “_”, habría que poner delante un carácter de escape. El carácter de escape lo que le indica a MySQL es: lo que viene justo detrás interpretarlo solo como un carácter. El carácter de escape por defecto es “\”.

Ejemplo: Supongamos que para la codificación de los artículos de una empresa se han usado todos los caracteres y queremos obtener un listado de los productos que empiezan por “P%XY”:

```
SELECT * FROM articulo WHERE nombre LIKE 'P\%XY%';
```

Expresiones regulares.

Una expresión regular describe un conjunto de cadenas por las que podríamos hacer un filtro. La expresión más simple o trivial no tendría caracteres especiales en ella y solo atendería a los caracteres que aparecen. Ej: “Hola” solo filtraría “Hola” y nada más, que puede aparecer en cualquier parte de la cadena/s a las que se le aplique el filtro.

Una expresión regular no trivial podría filtrar más de una cadena. Ej; “hola|adios”, nos permitiría filtrar todas aquellas cadenas que contengan “hola” o “adios”. En este ejemplo “|” es un carácter especial que acepta alguna de las cadenas que aparezcan al lado.

Para construir expresiones regulares debemos conocer los caracteres especial que están permitidos y que sentido tienen. Para una información más completa y detallada puedes consultar la siguiente web: International Components for Unicode website.

Algunos de los caracteres especiales:

- ^ El emparejamiento lo hará al principio de la cadena.
- $ El emparejamiento lo hará al final de la cadena.
- . Representa cualquier carácter.
- * Cualquier número de ocurrencias, incluido 0, del carácter o cadena que le preceda. Se ha de usar paréntesis para que “*” afecte a más de un carácter.
- + Cualquier número de ocurrencias a partir de uno. Similar a “*”.
- ? Cero o una ocurrencia del carácter o cadena que le preceda.
- | Permite poner filtros alternativos.
- {n} El carácter o patrón definido justo antes ha de repetirse “n” veces.
- {n,m} El carácter o patrón definido justo antes ha de repetirse entre “n” y “m” veces.

- {n,} El carácter o patrón definido justo antes se puede repetir “n” o más veces.
- [] Permite definir un rango de caracteres. Ej: [abc] validará si aparece la letra “a”, “b” o “c”. Si se ponen dos caracteres separados por un guión afecta a todos ellos, ej: [a-f] cualquier letra entre la “a” y la “f”; [0-9] cualquier número.

  Sin embargo si dentro del corchete, lo primero que aparece es “^”, significa negar lo anterior. Es decir, que no aparezcan.

Ejemplos: (el valor 0 es un resultado de falso y el valor uno sería un valor de verdadero, a la hora de usarlo en un SELECT). No se distinguen entre mayúsculas y minúsculas.

```
mysql>SELECT “En un lugar de la mancha de cuyo nombres..” REGEXP “ancha”;          ----  1
mysql>SELECT “En un lugar de la mancha de cuyo nombres..” REGEXP “ancla”;          ----  0
mysql>SELECT “En un lugar de la mancha de cuyo nombres..” REGEXP “^ancha”;         ----  0
mysql>SELECT “En un lugar de la mancha de cuyo nombres..” REGEXP “^en un”;         ----  1
mysql>SELECT “En un lugar de la mancha de cuyo nombres..” REGEXP “mbres..$”;       ----  1
mysql>SELECT “En un lugar de la mancha de cuyo nombres..” REGEXP “mbres$”;         ----  0
mysql>SELECT “hola” REGEXP “^….”;                                                  ----  1
mysql>SELECT “hola y adios” REGEXP “^….”;                                          ----  1
mysql>SELECT “hola” REGEXP “^….$”;                                                 ----  1
mysql>SELECT “hola y adios” REGEXP “^….$”;                                         ----  0
mysql>SELECT “hola” REGEXP “^…a*”;                                                 ----  1
mysql>SELECT “hol” REGEXP “^…a*”;                                                  ----  1
mysql>SELECT “holaaa” REGEXP “^…a*”;                                               ----  1
mysql>SELECT “holaholacolatoca” REGEXP “^(…a)*”;  /* lo acepta todo */            ----  1
mysql>SELECT “holaholacolatoca” REGEXP “^(…a)*$”;                                  ----  1
mysql>SELECT “holaholxcolxtoca” REGEXP “^(…a)*$”;                                  ----  0
mysql>SELECT “hola” REGEXP “^ho+la$”;                                              ----  1
mysql>SELECT “hoooola” REGEXP “^ho+la”;                                            ----  1
mysql>SELECT “hla” REGEXP “^ho+la”;                                                ----  0
mysql>SELECT “hla” REGEXP “^ho?la”;                                                ----  1
mysql>SELECT “hola” REGEXP “^ho?la”;                                               ----  1
mysql>SELECT “hoola” REGEXP “^ho?la”;                                              ----  0
mysql>SELECT “Dios te da nueces, pero no las parte” REGEXP “nueces|almendras”;     ----  1
mysql>SELECT “a Dios rogando y con el mazo dando” REGEXP “nueces|almendras”;       ----  0
mysql>SELECT “A quien lucha y suda, la suerte le ayuda” REGEXP “azar|suerte”;      ----  1
mysql>SELECT “45” REGEXP “^[0-9]{2}”;                                              ----  1
mysql>SELECT “145” REGEXP “^[0-9]{2}”;                                             ----  0
mysql>SELECT “5” REGEXP “^[0-9]{2}”;                                               ----  0
mysql>SELECT “2definicion” REGEXP “^[^0-9]”;                                       ----  0
```

A partir de la versión MySQL 3.23.4 no distingue entre mayúsculas ni minúsculas de la cadena del patrón con la cadena a comparar. Pero si necesitamos que distinga entre mayúsculas y minúsculas se pone BINARY.

```
mysql> select * from alumno where nombre regexp "^j";
+----+--------+-----------+-----------+-----------------+--------------+-----------+
| id | nombre | apellido1 | apellido2 | fecha_nacimiento | es_repetidor | telefono  |
+----+--------+-----------+-----------+-----------------+--------------+-----------+
|  2 | Juan   | Saez      | Vega      | 1998-04-02       | no           | 618253876 |
| 13 | Juan   | Sanchez   | Acedo     | 2000-12-20       | si           | 251333444 |
+----+--------+-----------+-----------+-----------------+--------------+-----------+
2 rows in set (0.00 sec)

mysql> select * from alumno where nombre regexp binary "^j";
Empty set (0.00 sec)

mysql> select * from alumno where nombre regexp binary "^J";
+----+--------+-----------+-----------+-----------------+--------------+-----------+
| id | nombre | apellido1 | apellido2 | fecha_nacimiento | es_repetidor | telefono  |
+----+--------+-----------+-----------+-----------------+--------------+-----------+
|  2 | Juan   | Saez      | Vega      | 1998-04-02       | no           | 618253876 |
| 13 | Juan   | Sanchez   | Acedo     | 2000-12-20       | si           | 251333444 |
+----+--------+-----------+-----------+-----------------+--------------+-----------+
2 rows in set (0.00 sec)
```

Operador IS.

Este operador nos permite comprobar si el valor de una determinada columna es NULL o no.

Ejemplo:

mysql> SELECT id, nombre, apellido1 FROM alumno WHERE apellido2 IS NULL;
mysql> SELECT id, nombre, apellido1, telefono FROM alumno
WHERE telefono IS NOT NULL;

```
mysql> SELECT id, nombre, apellido1 FROM alumno WHERE apellido2 IS NULL;
+----+--------+-----------+
| id | nombre | apellido1 |
+----+--------+-----------+
|  1 | Maria  | Sanchez   |
+----+--------+-----------+
1 row in set (0.00 sec)

mysql> SELECT id, nombre, apellido1, telefono FROM alumno
    ->         WHERE telefono IS NOT NULL;
+----+----------+-----------+-----------+
| id | nombre   | apellido1 | telefono  |
+----+----------+-----------+-----------+
|  2 | Juan     | Saez      | 618253876 |
|  4 | Lucia    | Sanchez   | 678516294 |
|  5 | Paco     | Martinez  | 692735409 |
|  7 | Cristina | Fernandez | 628349590 |
|  8 | Antonio  | Carretero | 612345633 |
| 11 | Pepe     | Sanchez   | 251222555 |
| 12 | Samuel   | Sanchez   | 251333777 |
| 13 | Juan     | Sanchez   | 251333444 |
| 14 | Zacarias | Sanchez   | 251333555 |
+----+----------+-----------+-----------+
9 rows in set (0.00 sec)
```

5.2. Funciones MySQL.

MySQL dispone de una serie de funciones que pueden ser de gran utilidad y que se pueden utilizar en la orden SELECT.

Estas funciones se agrupan en:

- Funciones de cadena.
- Funciones matemáticas.
- Funciones de fecha y hora.

5.2.1. Funciones de cadena.

Función.	**Descripción.**
CONCAT(cad1, cad2,..)	Devuelve la concatenación de todas las cadenas pasadas como parámetros "cad1, cad2,.." en el mismo orden en el que se pasaron. Si alguna cadena es NULL dará como resultado NULL.
CONCAT_WS(separador, cad1, cad2,..)	Devuelve la concatenación de todas las cadenas pasadas como parámetros "cad1, cad2,.." en el mismo orden en el que se pasaron e intercala entre ellas el parámetro "separador". En caso de que alguna cadena fuese NULL, será ignorada.
LOWER(cad)	Convierte a minúsculas todos los caracteres de la cadena.
UPPER(cad)	Convierte a mayúsculas todos los caracteres de la cadena.
SUBSTR(cad, posicion, longitud)	Devuelve una subcadena de longitud "longitud" empezando a partir de la posición "posicion" (incluida) de la cadena "cad". Sinónimos: MID y SUBSTRING.
LEFT(cad, numero)	Devuelve una subcadena formada por las primeras "numero" de caracteres de "cad" (empieza a contar por la izquierda).
RIGHT(cad, numero)	Devuelve una subcadena formada por las últimos "numero" de caracteres de "cad" (empieza a contar por la derecha).
LOCATE(subcad, cad, inicio)	Devuelve la primera vez que aparece la subcadena "subcad" dentro de la cadena "cad" a partir de la posición "inicio".
REVERSE(cad)	Devuelve la cadena inversa a la que se pasa como parámetro.
LTRIM(cad)	Elimina los caracteres en blanco a la izquierda de la cadena.
RTRIM(cad)	Elimina los caracteres en blanco a la derecha de la cadena.
TRIM(cad)	Elimina los caracteres en blanco a la derecha y a la izquierda.
LENGTH(cad)	Devuelve la longitud de la cadena.

5.2.2. Funciones matemáticas.

Función.	**Descripción.**
ABS(num)	Devuelve el valor absoluto de "num".
POW(x,y)	Devuelve el valor de "x" elevado a "y".
SQRT(num)	Devuelve la raíz cuadrada de "num".
PI()	Devuelve el valor del número PI.
ROUND(num)	Devuelve un número entero resultado de redondear "num". A partir de 0,5 considera la siguiente unidad.

TRUNCATE(num, dec) Devuelve el resultado de truncar “num” con “dec” decimales.

CEIL(num) Devuelve el entero inmediatamente superior.

FLOOR(num) Devuelve el entero inmediatamente inferior. Igual que TRUNCATE(num,0).

5.2.3. Funciones de fecha y hora.

Función. **Descripción.**

NOW() Devuelve la fecha y hora actual en formato: YYYY-MM-DD HH:MM:SS.

CURTIME() Devuelve la hora actual en formato: HH:MM:SS.

DATE(exp) Devuelve la fecha de una expresión de tiempo: YYYY-MM-DD HH:MM:SS.

DATEDIFF(f1,f2) Devuelve el número de días que hay entre las fechas: f1 y f2 (f1>f2, de lo contrario daría resultado negativo).

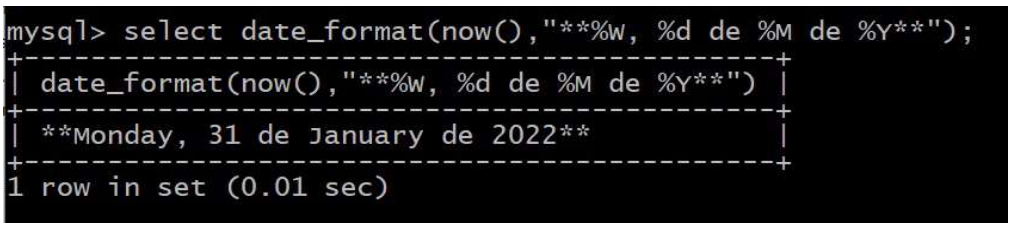
ADDDATE(fech, d) Suma un número de días “d” a la fecha “fech” y devuelve la fecha resultante. Ej: ADDDATE(“2022/01/25”,8) devolverá: “2022/02/02”.

DATE_FORMAT(fech, form) Devolverá la fecha dada “fech” con un formato específico “form”.

Los formatos disponibles son:

“%W”	Nombre del día.
“%D”	Cardinal del día.
“%d”	Día con dos dígitos.
“%e”	Día con uno o dos dígitos, según necesite.
“%Y”	Año con cuatro dígitos.
“%y”	Año con 2 dígitos.
“%M”	Nombre de mes completo.
“%m”	Número del mes con dos dígitos.
“%b”	Nombre del mes, pero solo las tres primeras letras.

Estos formatos se pueden incluir más de uno de ellos en la misma orden incluso usar otros caracteres. Ejemplo: supongamos que queremos mostrar la fecha de hoy con el formato:**lunes, 31 de enero de 2021**



Observa que los nombre de día y mes aparecen en inglés. Nomenclatura por defecto.

YEAR(fech) Devuelve el año con 4 dígitos de la fecha “fech”. Similar al formato “%Y”.

MONTH(fech) Devuelve el mes con 1 ó 2 dígitos, según necesite, de la fecha “fech”. Similar al formato “%m”.

MONTHNAME(fech) Devuelve el nombre del mes de la fecha “fech”. Similar al formato “%M”.

DAY(fech)	Devuelve el número del día con 1 ó dos dígitos, según necesite, de la fecha “fech”. Similar al formato “%e”.
DAYNAME(exp)	Devuelve el nombre del día de la expresión temporal “exp”. Similar a “%W”.
HOUR(exp)	Devuelve la hora del día de la expresión temporal “exp”. Similar a “%H” ó “%h”.
MINUTE(exp)	Devuelve el minuto de la expresión temporal “exp”. Similar a “%i”.
SECOND(exp)	Devuelve el segundo de la expresión temporal “exp”. Similar a “%S” ó “%s”.

Alguna combinación interesante:

TIME (NOW())	Devuelve la hora. Similar a CURTIME().
DATE (NOW())	Devuelve la fecha.

Establecer nomenclatura de nombres en español.

Por defecto los nombre de días y meses aparecen en inglés, configuración por defecto. Esto afecta a los resultados de las funciones: DATE_FORMAT, DAYNAME y MOTHNAME. Esta configuración se gestiona desde la variable global: lc_time_names, cuyo valor por defecto es: “en_US”. Para que aparezcan los nombres en castellano habría que cambiarla a: “en_ES”.

>SELECT @@lc_time_names; # para consultar el valor actual.
>SET lc_time_names = “en_ES”; # para asignarla un nuevo valor.

```
mysql>
mysql> select date_format(now(),"**%W, %d de %M de %Y**");
+-------------------------------------------+
| date_format(now(),"**%W, %d de %M de %Y**") |
+-------------------------------------------+
| **Monday, 31 de January de 2022**         |
+-------------------------------------------+
1 row in set (0.00 sec)

mysql> select @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| en_US           |
+-----------------+
1 row in set (0.00 sec)

mysql> set lc_time_names = "es_ES";
Query OK, 0 rows affected (0.00 sec)

mysql> select @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| es_ES           |
+-----------------+
1 row in set (0.00 sec)

mysql> select date_format(now(),"**%W, %d de %M de %Y**");
+-------------------------------------------+
| date_format(now(),"**%W, %d de %M de %Y**") |
+-------------------------------------------+
| **lunes, 31 de enero de 2022**            |
+-------------------------------------------+
1 row in set (0.00 sec)
```

También puede ser necesario modificar las variables globales que gestionan la zona horaria, para ajustar los resultados que ofrecen las funciones relacionadas con la hora a los de nuestra zona horaria. Nuestra zona horaria es: “Europe/Madrid” y las variable que lo gestiona: time_zone.

```
mysql> SELECT @@GLOBAL.time_zone, @@SESSION.time_zone;
mysql> SET GLOBAL time_zone = “Europe/Madrid”;
mysql> SET time_zone = “Europe/Madrid”;
```

Por defecto tiene el valor “SYSTEM”, toma la zona horaria del sistema, por lo que no ha de haber problemas.

Ejercicios.

Se utilizarán las bases de datos: institutodb, veterinariodb y contactosdb.

1. De la base de datos institutodb:
   a) Eliminar el segundo apellido del primer alumno. (..set apellido2= NULL..).
   b) Mostrar un listado en el que aparezcan dos columnas: id y “apellido1 apellido2, nombre”. En cabecera ha de aparecer: Código y Nombre de alumno.
2. Queremos asignar una cuenta de correo a los alumnos que estará compuesta por: letra inicial, tres primeras letras del primer apellido, tres primeras letras del segundo apellido y año de nacimiento (unidad y decena). El dominio será: politecnico.com

   Se ha de mostrar tres columnas: id, nombre completo (apellido1,apellido2, “, “ , nombre) y correo. En las cabeceras ha de aparecer: Código, Nombre, Correo electrónico.
3. En la tabla amo (veterinariodb) aparece el atributo: “apellidos”.

   Sobre esa tabla haz una consulta en al que aparezcan tres columnas: primer apellido, segundo apellido y los dos apellidos juntos (comprueba los resultados). Puede haber dueños de mascotas que solo tengan un apellido o un número indeterminado de espacios en blanco al principio, final o enmedio.
4. Calcular la circunferencia de un círculo de radio 5 cm. (circunferencia = 2 * pi * radio) y mostrarla en los siguientes formatos:
   a) Sin truncar ni redondear,
   b) Redondeando sin decimales.
   c) Redondear con dos decimales.
5. ¿Cuántos bytes tiene un KB? ¿Y un MB?
6. Mostrar la hora actual mostrando 4 columnas:
   a) Hora completa: hh:mm:ss
   b) Solo la hora.
   c) Solo los minutos.
   d) Solo segundos.
7. De la tabla alumnos queremos obtener un listado de las onomásticas ordenado. Realízalo también sobre la tabla contactos.

8. Mostrar un listado de los alumnos en el que aparezcan tres columnas: id, nombre completo (apellido1, apellido2, nombre) y es_repetidor. Estará ordenada por: el campo es_repetidor y por nombre completo (apellido1, apellido2, nombre). Ambos criterios ascendentes.

   Observa cuidadosamente los resultados y explica la ordenación.

9. Mostrar los alumnos del instituto, pero que solo aparezcan desde el quinto al noveno.

10. Realiza una consulta de todos los alumnos repetidores.

11. Realiza una consulta de todos los alumnos repetidores nacidos entre el año 1995 y el 2000.

12. Realiza una consulta de los no repetidores nacidos después de 1998.

13. Realiza una consulta de alumnos que nacieron en 1998, y otra de los que no nacieron ese año.

14. Consulta los alumnos que tienen un identificador entre 4 y 9.

15. Consulta los alumnos mayores de edad y menores de 25 que no tienen teléfono.

16. Listado de todos los alumnos con su edad actual. Para el cálculo considera que todos los años tienen 365.242 días (365 días, 5 horas y 48 minutos).

17. Listado de todos los alumnos cuyo segundo apellido termine en “ez”.

18. Listado en dos columnas ordenado alfabéticamente por apellidos y nombre. En la que aparezca:
    a) Nombre completo: Nombre (Primera letra mayúscula y resto minúscula), una “,” seguida de los apellidos en mayúsculas.
    b) Número de caracteres que tiene el nombre completo.

19. Expresar la fecha de hoy en el formato: **lunes, 31 de enero de 2021**.

20. De la base de datos de contactos, obtener un listado de las personas en el que aparezcan las siguientes columnas:
    a) Nombre.
    b) Apellidos.
    c) Nombre del mes de nacimiento.
    d) Día de nacimiento.
    e) Año de nacimiento.

    El objetivo es tener una lista de los próximos cumpleaños, por lo que se ha de ordenar por: mes, día y año. Para verlo cómodamente se han de mostrar de 15 en 15.

21. Listado de onomásticas de los contactos ordenado alfabéticamente y de 15 en 15.
22. Establecer la expresión regular que sea verdadera con los siguientes formatos:
    a) Código postal: cinco números.
    b) Número de DNI: siete u ocho números.
    c) Teléfono: 9 números.
    d) Nif: siete u ocho números seguidos o no de un guion y un letra.
    e) Matrícula de un coche, formato nuevo: 4 números, un guión y tres letras.

f) Matrícula de un coche, formato antiguo: una o dos letras (según provincia), guion, cuatro números, guion y una o dos letras. Guiones opcionales.
g) Matrícula de un coche, formato antiquísimo: una o dos letras (según provincia), un guion y entre un número y seis.