

Consultas SQL sobre una tabla

Apuntes BD

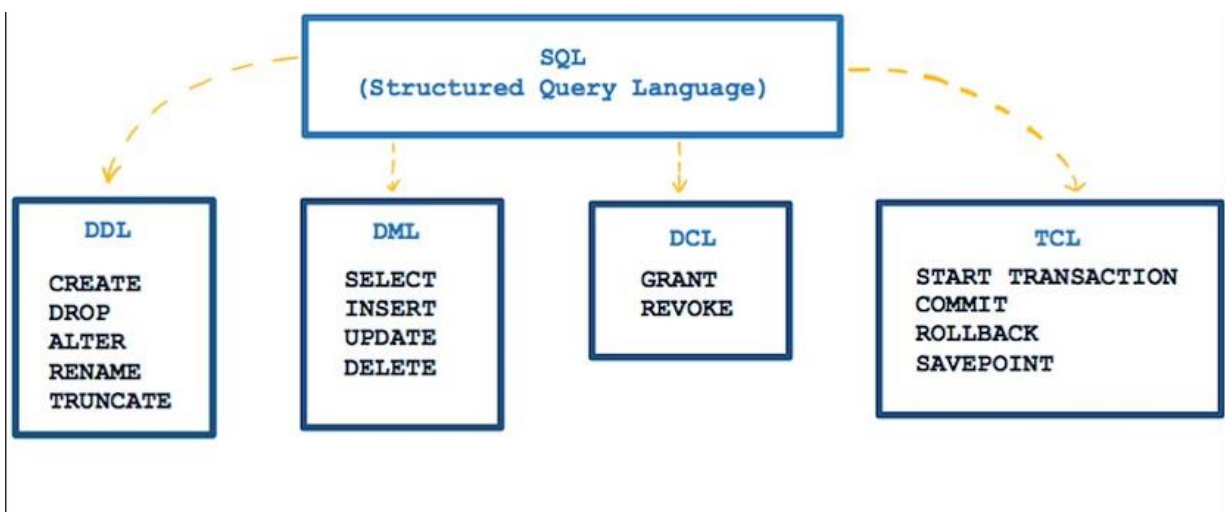
Índice

1	Realización de consultas SQL	1
1.1	El lenguaje DML de SQL	1
2	Consultas básicas sobre una tabla	2
2.1	Sintaxis de la instrucción SELECT	2
2.2	Cláusula SELECT	3
2.2.1	Cómo obtener los datos de todas las columnas de una tabla (SELECT *)	3
2.2.2	Cómo obtener los datos de algunas columnas de una tabla	5
2.2.3	Cómo realizar comentarios en sentencias SQL	7
2.2.4	Cómo obtener columnas calculadas.....	7
2.2.5	Cómo realizar alias de columnas con AS	9
2.2.6	Cómo utilizar funciones de MySQL en la cláusula SELECT	10
2.3	Modificadores ALL, DISTINCT y DISTINCTROW.....	12
2.4	Cláusula ORDER BY.....	14
2.4.1	Cómo ordenar de forma ascendente.....	14
2.4.2	Cómo ordenar de forma descendente.....	15
2.4.3	Cómo ordenar utilizando múltiples columnas.....	16
2.5	Cláusula LIMIT.....	16
2.6	Cláusula WHERE.....	17
2.6.1	Operadores disponibles en MySQL	18
2.6.2	Operador BETWEEN.....	20
2.6.3	Operador IN.....	20
2.6.4	Operador LIKE.....	20
2.6.5	Operador REGEXP y expresiones regulares.....	21
2.6.6	Operadores IS e IS NOT.....	22
2.7	Funciones disponibles en MySQL	22
2.7.1	Funciones con cadenas.....	22
2.7.2	Funciones de fecha y hora	24
2.7.3	Funciones matemáticas	26
2.7.4	Funciones para realizar búsquedas de tipo FULLTEXT.....	27
2.7.5	Funciones de control de flujo	28
3	Errores comunes	30
3.1	Error al comprobar si una columna es NULL	30
3.2	Error al comparar cadenas con patrones utilizando el operador =.....	30
3.3	Error al comparar cadenas con patrones utilizando el operador !=	30

3.4	Error al comparar un rango de valores con AND	31
3.5	Errores de conversión de tipos en la evaluación de expresiones.....	31

Realización de consultas SQL

1.1 El lenguaje DML de SQL



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

2.1 Sintaxis de la instrucción SELECT

Según la [documentación oficial de MySQL](#) ésta sería la sintaxis para realizar una consulta con la sentencia **SELECT** en MySQL:

```
1  SELECT
2    [ALL | DISTINCT | DISTINCTROW ]
3    [HIGH_PRIORITY]
4    [STRAIGHT_JOIN]
5    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
6    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
7    select_expr [, select_expr ...]
8    [FROM table_references]
9    [PARTITION partition_list]
10   [WHERE where_condition]
11   [GROUP BY {col_name | expr | position}
12    [ASC | DESC], ... [WITH ROLLUP]]
13   [HAVING having_condition]
14   [ORDER BY {col_name | expr | position}
15    [ASC | DESC], ...]
16   [LIMIT {[offset,] row_COUNT | row_COUNT OFFSET offset}]
17   [PROCEDURE procedure_name(argument_list)]
18   [INTO OUTFILE 'file_name'
19    [CHARACTER SET charset_name]
20    export_options
21    | INTO DUMPFILE 'file_name'
22    | INTO var_name [, var_name]]
23   [FOR UPDATE | LOCK IN SHARE MODE]]
```

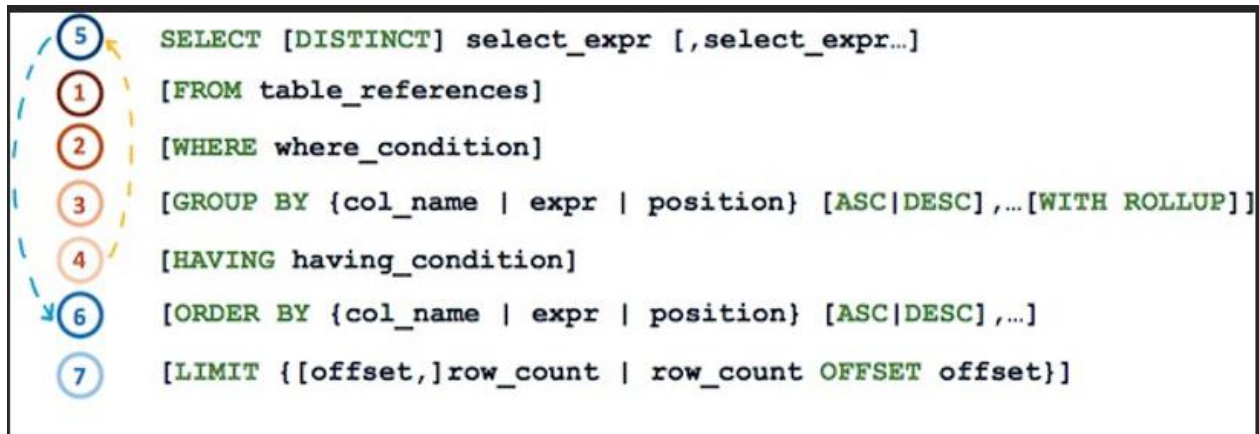
Para empezar con consultas sencillas podemos simplificar la definición anterior y quedarnos con la siguiente:

```
1  SELECT [DISTINCT] select_expr [, select_expr ...]
2  [FROM table_references]
3  [WHERE where_condition]
4  [GROUP BY {col_name | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
5  [HAVING having_condition]
6  [ORDER BY {col_name | expr | position} [ASC | DESC], ...]
7  [LIMIT {[offset,] row_COUNT | row_COUNT OFFSET offset}]
```

Es muy importante conocer **en qué orden se ejecuta cada una de las cláusulas** que forman la sentencia **SELECT**. El orden de ejecución es el siguiente:

- 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, que puede tener una o varias columnas y ninguna, una o varias filas.

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.

2.2 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:

- El **nombre de una columna** de la tabla sobre la que estamos realizando la consulta. Será una columna de la tabla que aparece en la cláusula **FROM**.
- Una **constante** que aparecerá en todas las filas de la tabla resultado.
- Una **expresión** que nos permite calcular nuevos valores.

2.2.1 Cómo obtener los datos de todas las columnas de una tabla (SELECT *)

Ejemplo:

Vamos a utilizar la siguiente base de datos de ejemplo para MySQL.

```

1 DROP DATABASE IF EXISTS instituto;
2 CREATE DATABASE instituto CHARACTER SET utf8mb4;
3 USE instituto;
4
5 CREATE TABLE alumno (
6   id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
7   nombre VARCHAR(100) NOT NULL,
8   apellido1 VARCHAR(100) NOT NULL,
9   apellido2 VARCHAR(100),
10  fecha_nacimiento DATE NOT NULL,
11  es_repetidor ENUM('sí', 'no') NOT NULL,
12  teléfono VARCHAR(9)
13 );
14
15 INSERT INTO alumno VALUES(1, 'María', 'Sánchez', 'Pérez', '1990-12-01', 'no',
16   NULL);
17 INSERT INTO alumno VALUES(2, 'Juan', 'Sáez', 'Vega', '1998-04-02', 'no',
18   618253876);
19 INSERT INTO alumno VALUES(3, 'Pepe', 'Ramírez', 'Gea', '1988-01-03', 'no', NULL
20   );
21 INSERT INTO alumno VALUES(4, 'Lucía', 'Sánchez', 'Ortega', '1993-06-13', 'sí',
22   678516294);
23 INSERT INTO alumno VALUES(5, 'Paco', 'Martínez', 'López', '1995-11-24', 'no',
24   692735409);
25 INSERT INTO alumno VALUES(6, 'Irene', 'Gutiérrez', 'Sánchez', '1991-03-28', 'sí
26   ', NULL);
27 INSERT INTO alumno VALUES(7, 'Cristina', 'Fernández', 'Ramírez', '1996-09-17',
28   'no', 628349590);
29 INSERT INTO alumno VALUES(8, 'Antonio', 'Carretero', 'Ortega', '1994-05-20', 's
30   í', 612345633);
31 INSERT INTO alumno VALUES(9, 'Manuel', 'Domínguez', 'Hernández', '1999-07-08',
32   'no', NULL);
33 INSERT INTO alumno VALUES(10, 'Daniel', 'Moreno', 'Ruiz', '1998-02-03', 'no',
34   NULL);

```

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

id	nombre	apellido1	apellido2	fecha_nacimiento	es_repetidor	téléfono
1	María	Sánchez	Pérez	1990/12/01	no	NULL
2	Juan	Sáez	Vega	1998/04/02	no	618253876
3	Pepe	Ramírez	Gea	1988/01/03	no	NULL
4	Lucía	Sánchez	Ortega	1993/06/13	sí	678516294
5	Paco	Martínez	López	1995/11/24	no	692735409
6	Irene	Gutiérrez	Sánchez	1991/03/28	sí	NULL
7	Cristina	Fernández	Ramírez	1996/09/17	no	628349590
8	Antonio	Carretero	Ortega	1994/05/20	sí	612345633

id	nombre	apellido1	apellido2	fecha_nacimiento	os_repetidor	teléfono
9	Manuel	Domínguez	Hernández	1999/07/08	no	NULL
10	Daniel	Moreno	Ruiz	1998/02/03	no	NULL

Vamos a ver qué consultas sería necesario realizar para obtener los siguientes datos.

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

```
1 SELECT *  
2 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 anterior de la siguiente forma obteniendo el mismo resultado:

```
1 select *  
2 from alumno;
```

Otra consideración que también debemos tener en cuenta es que una consulta SQL se puede escribir en una o varias líneas. Por ejemplo, la siguiente sentencia tendría el mismo resultado que la anterior:

```
1 SELECT * FROM alumno;
```

A lo largo del curso vamos a considerar como una buena práctica escribir las consultas SQL en varias líneas, empezando cada línea con la palabra reservada de la cláusula correspondiente que forma la consulta.

2.2.2 Cómo obtener los datos de algunas columnas de una tabla

2. Obtener el nombre de todos los alumnos.

```
1 SELECT nombre  
2 FROM alumno;
```

nombre

María

Juan

Pepe

Lucía

Paco

Irene

nombre

Cristina

Antonio

Manuel

Daniel

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

```
1 SELECT nombre, apellido1, apellido2
2 FROM alumno;
```

nombre	apellido1	apellido2
María	Sánchez	Pérez
Juan	Sáez	Vega
Pepe	Ramírez	Gea
Lucía	Sánchez	Ortega
Paco	Martínez	López
Irene	Gutiérrez	Sánchez
Cristina	Fernández	Ramírez
Antonio	Carretero	Ortega
Manuel	Domínguez	Hernández
Daniel	Moreno	Ruiz

Tenga en cuenta que el resultado de la consulta SQL mostrará las columnas que haya solicitado, siguiendo el orden en el que se hayan indicado. Por lo tanto la siguiente consulta:

```
1 SELECT apellido1, apellido2, nombre
2 FROM alumno;
```

Devolverá lo siguiente:

apellido1	apellido2	nombre
Sánchez	Pérez	María
Sáez	Vega	Juan
Ramírez	Gea	Pepe
Sánchez	Ortega	Lucía

apellido1	apellido2	nombre
Martínez	López	Paco
Gutiérrez	Sánchez	Irene
Fernández	Ramírez	Cristina
Carretero	Ortega	Antonio
Domínguez	Hernández	Manuel
Moreno	Ruiz	Daniel

2.2.3 Cómo realizar comentarios en sentencias SQL

Para escribir comentarios en nuestras sentencias SQL podemos hacerlo de diferentes formas.

```
1 -- Esto es un comentario
2 SELECT nombre, apellido1, apellido2 -- Esto es otro comentario
3 FROM alumno;
```

```
1 /* Esto es un comentario
2    de varias líneas */
3 SELECT nombre, apellido1, apellido2 /* Esto es otro comentario */
4 FROM alumno;
```

```
1 # Esto es un comentario
2 SELECT nombre, apellido1, apellido2 # Esto es otro comentario
3 FROM alumno;
```

2.2.4 Cómo obtener columnas calculadas

Ejemplo

Vamos a utilizar la siguiente base de datos de ejemplo para MySQL.

```
1 DROP DATABASE IF EXISTS tienda;
2 CREATE DATABASE tienda CHARACTER SET utf8mb4;
3 USE tienda;
4
5 CREATE TABLE ventas (
6     id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
7     cantidad_comprada INT UNSIGNED NOT NULL,
8     precio_por_elemento DECIMAL(7,2) NOT NULL
9 );
10
11 INSERT INTO ventas VALUES (1, 2, 1.50);
12 INSERT INTO ventas VALUES (2, 5, 1.75);
13 INSERT INTO ventas VALUES (3, 7, 2.00);
14 INSERT INTO ventas VALUES (4, 9, 3.50);
15 INSERT INTO ventas VALUES (5, 6, 9.99);
```

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

	id	cantidad_comprada	precio_por_elemento
1	2		1.50
2	5		1.75
3	7		2.00
4	9		3.50
5	6		9.99

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:

```
1 SELECT id, cantidad_comprada, precio_por_elemento, cantidad_comprada *  
   precio_por_elemento  
2 FROM ventas;
```

Y el resultado sería el siguiente:

id	cantidad_comprada	precio_por_elemento	cantidad_comprada * precio_por_elemento
1	2	1.50	3.00
2	5	1.75	8.75
3	7	2.00	14.00
4	9	3.50	31.50
5	6	9.99	59.94

Ejemplo

Vamos a utilizar la siguiente base de datos de ejemplo para MySQL.

```
1 DROP DATABASE IF EXISTS company;  
2 CREATE DATABASE company CHARACTER SET utf8mb4;  
3 USE company;  
4  
5 CREATE TABLE offices (  
6   office INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
7   city VARCHAR(50) NOT NULL,  
8   region VARCHAR(50) NOT NULL,  
9   manager INT UNSIGNED,  
10  target DECIMAL(9,2) NOT NULL,  
11  sales DECIMAL(9,2) NOT NULL
```

```

12 );
13
14 INSERT INTO offices VALUES (11, 'New York', 'Eastern', 106, 575000, 692637);
15 INSERT INTO offices VALUES (12, 'Chicago', 'Eastern', 104, 800000, 735042);
16 INSERT INTO offices VALUES (13, 'Atlanta', 'Eastern', NULL, 350000, 367911);
17 INSERT INTO offices VALUES (21, 'Los Angeles', 'Western', 108, 725000, 835915);
18 INSERT INTO offices VALUES (22, 'Denver', 'Western', 108, 300000, 186042);

```

Supongamos que tenemos una tabla llamada `oficinas` que contiene información sobre las ventas reales que ha generado y el valor de ventas esperado, y nos gustaría conocer si las oficinas han conseguido el objetivo propuesto, y si están por debajo o por encima del valor de ventas esperado.

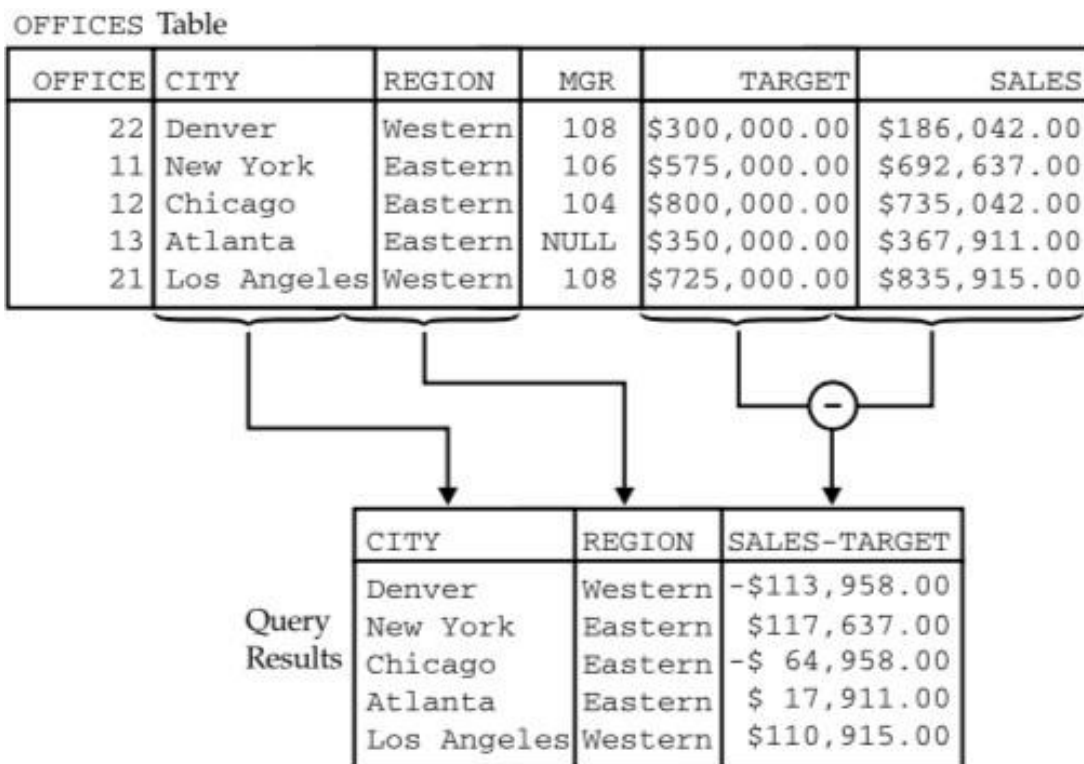


Imagen: Imagen extraída del libro *SQL: The Complete Reference* de James R. Groff y otros.

En este caso podríamos realizar la siguiente consulta:

```

1 SELECT city, region, sales - target
2 FROM offices;

```

2.2.5 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:

```
1 SELECT id, cantidad_comprada, precio_por_elemento, cantidad_comprada *  
   precio_por_elemento AS 'precio total'  
2 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:

	id	cantidad_comprada	precio_por_elemento	precio total
1	2		1.50	3.00
2	5		1.75	8.75
3	7		2.00	14.00

2.2.6 Cómo utilizar funciones de MySQL en la cláusula `SELECT`

Es posible hacer uso de funciones específicas de MySQL en la cláusula `SELECT`. MySQL nos ofrece funciones matemáticas, funciones para trabajar con cadenas y funciones para trabajar con fechas y horas. Algunos ejemplos de las funciones de MySQL que utilizaremos a lo largo del curso son las siguientes.

Funciones con cadenas

Función	Descripción
<code>CONCAT</code>	Concatena cadenas
<code>CONCAT_WS</code>	Concatena cadenas con un separador
<code>LOWER</code>	Devuelve una cadena en minúscula
<code>UPPER</code>	Devuelve una cadena en mayúscula
<code>SUBSTR</code>	Devuelve una subcadena

Funciones matemáticas

Función	Descripción
<code>ABS()</code>	Devuelve el valor absoluto
<code>POW(x, y)</code>	Devuelve el valor de x elevado a y
<code>SQRT()</code>	Devuelve la raíz cuadrada
<code>PI()</code>	Devuelve el valor del número <code>PI</code>
<code>ROUND()</code>	Redondea un valor numérico

Función	Descripción
<code>TRUNCATE()</code>	Trunca un valor numérico

Funciones de fecha y hora

Función	Descripción
<code>NOW()</code>	Devuelve la fecha y la hora actual
<code>CURTIME()</code>	Devuelve la hora actual

En la documentación oficial puede encontrar la [referencia completa de todas las funciones y operadores disponibles en MySQL](#).

Ejemplo

Obtener el nombre y los apellidos de todos los alumnos en una única columna.

```
1 SELECT CONCAT(nombre, apellido1, apellido2) AS nombre_completo
2 FROM alumno;
```

nombre_completo
MaríaSánchezPérez
JuanSáezVega
PepeRamírezGea
LucíaSánchezOrtega
PacoMartínezLópez
IreneGutiérrezSánchez
CristinaFernándezRamírez
AntonioCarreteroOrtega
ManuelDomínguezHernández
DanielMorenoRuiz

En este caso estamos haciendo uso de la [función CONCAT de MySQL](#) y la palabra reservada `AS` para crear un **alias de la columna** y renombrarla como *nombre_completo*.

La [función CONCAT de MySQL](#) no añade ningún espacio entre las columnas, por eso los valores de las tres columnas aparecen como una sola cadena sin espacios entre ellas. Para resolver este problema podemos hacer uso de la [función CONCAT_WS](#) que nos permite definir un carácter separador entre cada columna. En el siguiente ejemplo haremos uso de la [función CONCAT_WS](#) y usaremos un espacio en blanco como separador.

```
1 SELECT CONCAT_WS(' ', nombre, apellido1, apellido2) AS nombre_completo
2 FROM alumno;
```

nombre_completo
María Sánchez Pérez
Juan Sáez Vega
Pepe Ramírez Gea
Lucía Sánchez Ortega
Paco Martínez López
Irene Gutiérrez Sánchez
Cristina Fernández Ramírez
Antonio Carretero Ortega
Manuel Domínguez Hernández
Daniel Moreno Ruiz

Importante: La función `CONCAT` devolverá `NULL` cuando alguna de las cadenas que está concatenando es igual `NULL`, mientras que la función `CONCAT_WS` omitirá todas las cadenas que sean igual a `NULL` y realizará la concatenación con el resto de cadenas.

Ejercicios

1. Obtener el nombre y los apellidos de todos los alumnos en una única columna en minúscula.
2. Obtener el nombre y los apellidos de todos los alumnos en una única columna en mayúscula.
3. Obtener el nombre y los apellidos de todos los alumnos en una única columna. Cuando el segundo apellido de un alumno sea `NULL` se devolverá el nombre y el primer apellido concatenados en mayúscula, y cuando no lo sea, se devolverá el nombre completo concatenado tal y como aparece en la tabla.

2.3 Modificadores `ALL`, `DISTINCT` y `DISTINCTROW`

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.
- `DISTINCTROW` es un sinónimo de `DISTINCT`.

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**.

```
1 SELECT apellido1
2 FROM alumno;
```

apellido1

Sánchez

Sáez

Ramírez

Sánchez

Martínez

Gutiérrez

Fernández

Carretero

Domínguez

Moreno

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

```
1 SELECT DISTINCT apellido1
2 FROM alumno;
```

apellido1

Sánchez

Sáez

Ramírez

Martínez

Gutiérrez

Fernández

Carretero

Domínguez

Moreno

Si en la cláusula **SELECT** utilizamos **DISTINCT** con más de una columna, la consulta seguirá eliminando todas las filas repetidas que existan. Por ejemplo, si tenemos las columnas **apellido1**, **apellido2** y **nombre**, se eliminarán todas las filas que tengan los mismos valores en las tres columnas.


```
1 SELECT DISTINCT apellido1, apellido2, nombre
2 FROM alumno;
```

apellido1

Sánchez

Sáez

Ramírez

Sánchez

Martínez

Gutiérrez

Fernández

Carretero

Domínguez

Moreno

En este ejemplo no se ha eliminado ninguna fila porque no existen alumnos que tengan los mismos apellidos y el mismo nombre.

2.4 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:

```
1 [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.

2.4.1 Cómo ordenar de forma ascendente

Ejemplo

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

```
1 SELECT apellido1, apellido2, nombre
2 FROM alumno
3 ORDER BY apellido1;
```

Si no indicamos nada en la cláusula **ORDER BY** se ordenará por defecto de forma ascendente.

La consulta anterior es equivalente a esta otra.

```
1 SELECT apellido1, apellido2, nombre
2 FROM alumno
3 ORDER BY apellido1 ASC;
```

El resultado de ambas consultas será:

nombre	apellido1	apellido2
Carretero	Ortega	Antonio
Domínguez	Hernández	Manuel
Fernández	Ramírez	Cristina
Gutiérrez	Sánchez	Irene
Martínez	López	Paco
Moreno	Ruiz	Daniel
Ramírez	Gea	Pepe
Sáez	Vega	Juan
Sánchez	Pérez	María
Sánchez	Ortega	Lucía

Las filas están ordenadas correctamente por el primer apellido, pero todavía hay que resolver cómo ordenar el listado cuando existen varias filas donde coincide el valor del primer apellido. En este caso tenemos dos filas donde el primer apellido es **Sánchez**:

nombre	apellido1	apellido2
...
Sánchez	Pérez	María
Sánchez	Ortega	Lucía

Más adelante veremos cómo podemos ordenar el listado teniendo en cuenta más de una columna.

2.4.2 Cómo ordenar de forma descendente

Ejemplo

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

```
1 SELECT apellido1, apellido2, nombre
2 FROM alumno
3 ORDER BY apellido1 DESC;
```

2.4.3 Cómo ordenar utilizando múltiples columnas

Ejemplo

En este ejemplo vamos obtener un listado de todos los alumnos ordenados por el primer apellido, segundo apellido y nombre, de forma ascendente.

```
1 SELECT apellido1, apellido2, nombre
2 FROM alumno
3 ORDER BY apellido1, apellido2, nombre;
```

En lugar de indicar el nombre de las columnas en la cláusula `ORDER BY` podemos indicar sobre la posición donde aparecen en la cláusula `SELECT`, de modo que la consulta anterior sería equivalente a la siguiente:

```
1 SELECT apellido1, apellido2, nombre
2 FROM alumno
3 ORDER BY 1, 2, 3;
```

2.5 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

```
1 [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`.

Ejemplo

```
1 DROP DATABASE IF EXISTS google;
2 CREATE DATABASE google CHARACTER SET utf8mb4;
3 USE google;
4
5 CREATE TABLE resultado (
6   id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
7   nombre VARCHAR(100) NOT NULL,
8   descripcion VARCHAR(200) NOT NULL,
9   url VARCHAR(512) NOT NULL
10 );
11
12 INSERT INTO resultado VALUES (1, 'Resultado 1', 'Descripción 1', 'http://.....');
```

```
13 INSERT INTO resultado VALUES (2, 'Resultado 2', 'Descripción 2', 'http://.....')
14 ;
15 INSERT INTO resultado VALUES (3, 'Resultado 3', 'Descripción 3', 'http://.....')
16 ;
17 INSERT INTO resultado VALUES (4, 'Resultado 4', 'Descripción 4', 'http://.....')
18 ;
19 INSERT INTO resultado VALUES (5, 'Resultado 5', 'Descripción 5', 'http://.....')
20 ;
21 INSERT INTO resultado VALUES (6, 'Resultado 6', 'Descripción 6', 'http://.....')
22 ;
23 INSERT INTO resultado VALUES (7, 'Resultado 7', 'Descripción 7', 'http://.....')
24 ;
25 INSERT INTO resultado VALUES (8, 'Resultado 8', 'Descripción 8', 'http://.....')
26 ;
27 INSERT INTO resultado VALUES (9, 'Resultado 9', 'Descripción 9', 'http://.....')
28 ;
29 INSERT INTO resultado VALUES (10, 'Resultado 10', 'Descripción 10', 'http
30 ://.....');
31 INSERT INTO resultado VALUES (11, 'Resultado 11', 'Descripción 11', 'http
32 ://.....');
33 INSERT INTO resultado VALUES (12, 'Resultado 12', 'Descripción 12', 'http
34 ://.....');
35 INSERT INTO resultado VALUES (13, 'Resultado 13', 'Descripción 13', 'http
36 ://.....');
37 INSERT INTO resultado VALUES (14, 'Resultado 14', 'Descripción 14', 'http
38 ://.....');
39 INSERT INTO resultado VALUES (15, 'Resultado 15', 'Descripción 15', 'http
40 ://.....');
```

1. Suponga que queremos mostrar en una página web las filas que están almacenadas en la tabla `resultados`, y queremos mostrar la información en diferentes páginas, donde cada una de las páginas muestra solamente 5 resultados. ¿Qué consulta SQL necesitamos realizar para incluir los primeros 5 resultados de la primera página?
2. ¿Qué consulta necesitaríamos para mostrar resultados de la segunda página?
3. ¿Y los resultados de la tercera página?

2.6 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`.

Podemos diferenciar cinco tipos de condiciones que pueden aparecer en la cláusula `WHERE`:

- Condiciones para comparar valores o expresiones.
- Condiciones para comprobar si un valor está dentro de un rango de valores.
- Condiciones para comprobar si un valor está dentro de un conjunto de valores.
- Condiciones para comparar cadenas con patrones.

- Condiciones para comprobar si una columna tiene valores a **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.

2.6.1 Operadores disponibles en MySQL

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

Operadores aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo

Operadores de comparación

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

Operadores lógicos

Operador	Descripción
AND	Y lógica
&&	Y lógica
OR	O lógica
 	O lógica

Operador	Descripción
NOT	Negación lógica
!	Negación lógica

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.

```
1 SELECT nombre
2 FROM alumno
3 WHERE apellido1 = 'Martínez';
```

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

```
1 SELECT *
2 FROM alumno
3 WHERE id = 9;
```

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

```
1 SELECT nombre, fecha_nacimiento
2 FROM alumno
3 WHERE fecha_nacimiento >= '1997/01/01';
```

2.6.2 Operador BETWEEN

Sintaxis:

```
1 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:

```
1 SELECT *
2 FROM pedido
3 WHERE fecha_pedido BETWEEN '2018-01-01' AND '2018-01-31';
```

2.6.3 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**.

```
1 SELECT *
2 FROM alumno
3 WHERE apellido1 IN ('Sánchez', 'Martínez', 'Domínguez');
```

Ejemplo:

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

```
1 SELECT *
2 FROM alumno
3 WHERE apellido1 NOT IN ('Sánchez', 'Martínez', 'Domínguez');
```

2.6.4 Operador LIKE

Sintaxis:

```
1 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.

```
1 SELECT *
2 FROM alumno
3 WHERE apellido1 LIKE 'S%';
```

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

```
1 SELECT *
2 FROM alumno
3 WHERE apellido1 LIKE '%Z';
```

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

```
1 SELECT *
2 FROM alumno
3 WHERE nombre LIKE '%a%';
```

Devuelva un listado de todos los alumnos que tengan un nombre de cuatro caracteres.

```
1 SELECT *
2 FROM alumno
3 WHERE nombre LIKE '____';
```

Devuelve un listado de todos los productos 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.

```
1 SELECT *
2 FROM producto
3 WHERE nombre LIKE 'A$%BC%' ESCAPE '$';
```

Por defecto, se utiliza el carácter \" como carácter de escape. De modo que podríamos escribir la consulta de la siguiente manera.

```
1 SELECT *
2 FROM producto
3 WHERE nombre LIKE 'A\\%BC%';
```

2.6.5 Operador REGEXP y expresiones regulares

El operador **REGEXP** nos permite realizar búsquedas mucho más potentes haciendo uso de expresiones regulares. Puede consultar la [documentación oficial](#) para conocer más detalles sobre cómo usar este operador.

2.6.6 Operadores 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**.

```
1 SELECT *
2 FROM alumno
3 WHERE teléfono IS NULL;
```

Ejemplo:

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

```
1 SELECT *
2 FROM alumno
3 WHERE teléfono IS NOT NULL;
```

2.7 Funciones disponibles en MySQL

A continuación se muestran algunas funciones disponibles en MySQL que pueden ser utilizadas para realizar consultas.

Las funciones se pueden utilizar en las cláusulas **SELECT**, **WHERE** y **ORDER BY**.

2.7.1 Funciones con cadenas

Función	Descripción
CONCAT	Concatena cadenas
CONCAT_WS	Concatena cadenas con un separador
LOWER	Devuelve una cadena en minúscula
UPPER	Devuelve una cadena en mayúscula
SUBSTR	Devuelve una subcadena
LEFT	Devuelve los caracteres de una cadena, empezando por la izquierda
RIGHT	Devuelve los caracteres de una cadena, empezando por la derecha
CHAR_LENGTH	Devuelve el número de caracteres que tiene una cadena
LENGTH	Devuelve el número de bytes que ocupa una cadena

Función	Descripción
<code>REVERSE</code>	Devuelve una cadena invirtiendo el orden de sus caracteres
<code>LTRIM</code>	Elimina los espacios en blanco que existan al inicio de una cadena
<code>RTRIM</code>	Elimina los espacios en blanco que existan al final de una cadena
<code>TRIM</code>	Elimina los espacios en blanco que existan al inicio y al final de una cadena
<code>REPLACE</code>	Permite reemplazar un carácter dentro de una cadena

En la documentación oficial puede encontrar la [referencia completa de todas las funciones disponibles en MySQL para trabajar con cadenas de caracteres](#).

2.7.2 Funciones de fecha y hora

Función	Descripción
<code>NOW()</code>	Devuelve la fecha y la hora actual
<code>CURTIME()</code>	Devuelve la hora actual
<code>ADDDATE</code>	Suma un número de días a una fecha y calcula la nueva fecha
<code>DATE_FORMAT</code>	Nos permite formatear fechas
<code>DATEDIFF</code>	Calcula el número de días que hay entre dos fechas
<code>YEAR</code>	Devuelve el año de una fecha
<code>MONTH</code>	Devuelve el mes de una fecha
<code>MONTHNAME</code>	Devuelve el nombre del mes de una fecha
<code>DAY</code>	Devuelve el día de una fecha
<code>DAYNAME</code>	Devuelve el nombre del día de una fecha
<code>hour</code>	Devuelve las horas de un valor de tipo <code>DATETIME</code>
<code>minute</code>	Devuelve los minutos de un valor de tipo <code>DATETIME</code>
<code>second</code>	Devuelve los segundos de un valor de tipo <code>DATETIME</code>

En la documentación oficial puede encontrar la [referencia completa de todas las funciones disponibles en MySQL para trabajar con fechas y horas](#).

Configuración regional en MySQL Server (*locale*)

Importante: Tenga en cuenta que para que los nombres de los meses y las abreviaciones aparezcan en español deberá configurar la variable del sistema `lc_time_names`. Esta variable afecta al resultado de las funciones `DATE_FORMAT`, `DAYNAME` y `MONTHNAME`.

En MySQL las variables se pueden definir como **variables globales** o **variables de sesión**. La diferencia que existe entre ellas es que una variable de sesión pierde su contenido cuando cerramos la sesión con el servidor, mientras que una variable global mantiene su valor hasta que se realiza un reinicio del servicio o se modifica por otro valor. **Las variables globales sólo pueden ser configuradas por usuarios con privilegios de administración.**

Para configurar la variable `lc_time_names` como una **variable global**, con la configuración regional de España tendrá que utilizar la palabra reservada `GLOBAL`, como se indica en el siguiente ejemplo.

```
1 SET GLOBAL lc_time_names = 'es_ES';
```

Para realizar la configuración como una **variable de sesión** tendría que ejecutar:

```
1 SET lc_time_names = 'es_ES';
```

Una vez hecho esto podrá consultar sus valores haciendo:

```
1 SELECT @@GLOBAL.time_zone, @@SESSION.time_zone;
```

En la documentación oficial pueden encontrar [más información sobre la configuración regional en MySQL Server](#).

Para consultar el valor de todas las **variables de sesión del sistema** podemos utilizar la sentencia:

```
1 SHOW VARIABLES;
```

O con el modificador `SESSION`:

```
1 SHOW SESSION VARIABLES;
```

Para consultar el valor de todas las **variables globales del sistema** podemos utilizar la sentencia:

```
1 SHOW GLOBAL VARIABLES;
```

Configuración de la zona horaria en MySQL Server (*time zone*)

Importante: Tenga en cuenta que también será necesario configurar nuestra zona horaria para que las funciones relacionadas con la hora devuelvan los valores de nuestra zona horaria. En este caso tendrá que configurar la variable del sistema `time_zone`, como **variable global** o como **variable de sesión**. A continuación, se muestra un ejemplo de cómo habría que configurar las variables para la zona horaria de Madrid.

```
1 SET GLOBAL time_zone = 'Europe/Madrid';
2 SET time_zone = 'Europe/Madrid';
```

Podemos comprobar el estado de las variables haciendo:

```
1 SELECT @@GLOBAL.time_zone, @@SESSION.time_zone;
```

En la documentación oficial pueden encontrar [más información sobre la configuración de la zona horaria en MySQL Server](#).

Ejemplos:

`NOW()` devuelve la fecha y la hora actual.

```
1 SELECT NOW();
```

`CURTIME()` devuelve la hora actual.

```
1 SELECT CURTIME();
```

2.7.3 Funciones matemáticas

Función	Descripción
<code>ABS()</code>	Devuelve el valor absoluto
<code>POW(x, y)</code>	Devuelve el valor de x elevado a y
<code>SQRT</code>	Devuelve la raíz cuadrada

Consultas SQL sobre una tabla

<code>PI()</code>	Devuelve el valor del número <code>PI</code>
<code>ROUND</code>	Redondea un valor numérico
<code>TRUNCATE</code>	Trunca un valor numérico
<code>CEIL</code>	Devuelve el entero inmediatamente superior o igual
<code>FLOOR</code>	Devuelve el entero inmediatamente inferior o igual
<code>MOD</code>	Devuelve el resto de una división

En la documentación oficial puede encontrar la [referencia completa de todas las funciones matemáticas disponibles en MySQL](#).

Ejemplos:

`ABS` devuelve el valor absoluto de un número.

```
1 SELECT ABS(-25);  
2  
3 25
```

`POW(x, y)` devuelve el valor de x elevado a y.

```
1 SELECT POW(2, 10);
2
3 1024
```

SQRT devuelve la raíz cuadrada de un número.

```
1 SELECT SQRT(1024);
2
3 32
```

PI() devuelve el valor del número **PI**.

```
1 SELECT PI();
2
3 3.141593
```

ROUND redondea un valor numérico

```
1 SELECT ROUND(37.6234);
2
3 38
```

TRUNCATE Trunca un valor numérico.

```
1 SELECT TRUNCATE(37.6234, 0);
2
3 37
```

CEIL devuelve el **entero inmediatamente superior o igual** al parámetro de entrada.

```
1 SELECT CEIL(4.3);
2
3 5
```

FLOOR devuelve el **entero inmediatamente inferior o igual** al parámetro de entrada.

```
1 SELECT FLOOR(4.3);
2
3 4
```

2.7.4 Funciones para realizar búsquedas de tipo FULLTEXT

Ejemplo

En este ejemplo vamos a crear un índice **FULLTEXT** sobre las columnas **nombre** y **descripcion** de la tabla **producto**, para permitir realizar búsquedas más eficientes sobre esas columnas.

```
1 CREATE FULLTEXT INDEX idx_nombre_descripcion ON producto(nombre, descripcion);
```

Una vez creado el índice ejecutamos la consulta haciendo uso de **MATCH** y **AGAINST**.

```
1 SELECT *
2 FROM producto
```

```
3 WHERE MATCH(nombre, descripcion) AGAINST ('acero');
```

Puede encontrar más información sobre la creación de índices en MySQL en la [documentación oficial](#).

2.7.5 Funciones de control de flujo

Función	Descripción
<code>CASE</code>	Operador para realizar condicionales múltiples
<code>IF()</code>	Permite realizar condiciones de tipo <code>IF/ELSE</code>
<code>IFNULL()</code>	Devuelve el primer argumento si no es <code>NULL</code> , en caso contrario devuelve el segundo argumento
<code>NULLIF()</code>	Devuelve <code>NULL</code> si los dos argumentos son iguales, en caso contrario devuelve el primer argumento

En la documentación oficial puede encontrar la [referencia completa de todas las funciones de control de flujo disponibles en MySQL](#).

Ejemplo del operador `CASE`:

Existen dos formas de utilizar el operador `CASE`:

Sintaxis 1:

```
1 CASE value
2   WHEN compare_value THEN result
3   [WHEN compare_value THEN result ...]
4   [ELSE result]
5 END
```

En este caso, junto al operador `CASE` se utiliza un valor, el nombre de una columna o una expresión y se compara con una serie de valores. Cuando se encuentra la primera coincidencia, se devuelve el resultado asociado y no se siguen evaluando el resto de comparaciones.

```
1 SELECT
2   nombre, apellido1, apellido2,
3   CASE es_repetidor
4     WHEN 'sí' THEN 'Repite'
5     WHEN 'no' THEN 'No repite'
6   END
7 FROM alumno;
```

Sintaxis 2:

```
1 CASE
2   WHEN condition THEN result
3   [WHEN condition THEN result ...]
4   [ELSE result]
5 END
```

En este caso se utiliza una condición en cada comparación y cuando se encuentra la primera condición que sea cierta, se devuelve el resultado asociado y no se siguen evaluando el resto de condiciones.

```
1 SELECT
2     producto, cantidad,
3     CASE
4         WHEN cantidad > 100 THEN 'Stock suficiente'
5         WHEN cantidad > 50 THEN 'Stock moderado'
6         ELSE 'Stock bajo'
7     END AS estado
8 FROM
9     producto;
```

Ejemplo de la función IF:

La sintaxis de la función IF es la siguiente:

```
1 IF(expr1,expr2,expr3)
```

Si la expresión `expr1` es `TRUE` entonces devuelve el valor de `expr2`, en caso contrario devuelve el valor de `expr3`.

```
1 SELECT
2     nombre, apellido1, apellido2, IF(es_repetidor = 'sí', 'Repite', 'No repite')
3 FROM alumno;
```

Ejemplo de la función IFNULL:

La sintaxis de la función IFNULL es la siguiente:

```
1 IFNULL(expr1,expr2)
```

Si la expresión `expr1` no es `NULL` entonces devuelve el valor de `expr1`, en caso contrario devuelve el valor de `expr2`.

```
1 SELECT
2     nombre, apellido1, apellido2, IFNULL(telefono, ' Teléfono no disponible')
3 FROM alumno;
```

Ejemplo de la función NULLIF:

La sintaxis de la función NULLIF es la siguiente:

```
1 NULLIF(expr1,expr2)
```

Esta función devuelve `NULL` si `expr1` es igual a `expr2`, en caso contrario devuelve el valor de `expr1`.

Errores comunes

3.1 Error al comprobar si una columna es NULL

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

Consulta incorrecta.

```
1 SELECT *
2 FROM alumno
3 WHERE teléfono = NULL;
```

Consulta correcta.

```
1 SELECT *
2 FROM alumno
3 WHERE teléfono IS NULL;
```

3.2 Error al comparar cadenas con patrones utilizando el operador =

Ejemplo: Devuelve un listado de los alumnos cuyo primer apellido empieza por `S`.

Consulta incorrecta.

```
1 SELECT *
2 FROM alumno
3 WHERE apellido1 = 'S%';
```

Consulta correcta.

```
1 SELECT *
2 FROM alumno
3 WHERE apellido1 LIKE 'S%';
```

3.3 Error al comparar cadenas con patrones utilizando el operador !=

Ejemplo: Devuelve un listado de los alumnos cuyo primer apellido no empieza por `S`.

Consulta incorrecta.

```
1 SELECT *
2 FROM alumno
3 WHERE apellido1 != 'S%';
```

Consulta correcta.

```
1 SELECT *
2 FROM alumno
3 WHERE apellido1 NOT LIKE 'S%';
```

3.4 Error al comparar un rango de valores con AND

Cuando queremos comparar si un valor está dentro de un rango tenemos que utilizar dos condiciones unidas con la operación lógica **AND**. En el siguiente ejemplo se muestra una consulta incorrecta donde la segunda condición siempre será verdadera porque no estamos comparando con ningún valor, estamos poniendo un valor constante que al ser distinto de 0 siempre nos dará un valor verdadero como resultado de la comparación.

Consulta incorrecta

```
1 SELECT *
2 FROM alumno
3 WHERE fecha_nacimiento >= '1999/01/01' AND '1999/12/31'
```

Consulta correcta

```
1 SELECT *
2 FROM alumno
3 WHERE fecha_nacimiento >= '1999/01/01' AND fecha_nacimiento <= '1999/12/31'
```

3.5 Errores de conversión de tipos en la evaluación de expresiones

Cuando se utiliza un operador con operandos de diferentes tipos de dato, MySQL realiza una conversión automática de tipo para que los operandos sean compatibles. Por ejemplo, convierte automáticamente cadenas en números según sea necesario y viceversa.

```
1 SELECT 1 + '1';
2 -> 2
```

```
1 SELECT CONCAT(1, '1');
2 -> '11'
```

Ejemplo: Resta entre dos valores de tipo **VARCHAR**

Las cadenas las convierte a datos de tipo entero de la mejor forma posible y luego realiza la resta. En este caso la cadena **'2021-01-31'** la convierte en el número entero 2021 y la cadena **'2021-02-01'** en el número entero 2021, por este motivo el resultado de la resta es 0.

```
1 SELECT '2021-01-31' - '2021-02-01';
2 -> 0
```

Ejemplo: Resta entre dos valores de tipo `DATE`

En este ejemplo estamos convirtiendo las cadenas a un valor de tipo `DATE` y luego se realiza la resta. En este caso, los datos de tipo `DATE` los convierte a datos de tipo entero pero de una forma más precisa que la conversión anterior, ya que la cadena `'2008-08-31'` la convierte en el número entero 20080831 y la cadena `'2008-09-01'` en el número entero 20080901, por este motivo el resultado de la resta es -70.

```
1 SELECT CAST('2021-01-31' AS DATE) - CAST('2021-02-01' AS DATE);
2 -> -70
```

Ejemplo: Resta entre dos valores de tipo `INT`

```
1 SELECT 20210131 - 20210201;
2 -> -70
```

Ejemplo: Resta entre dos valores de tipo `DATE` utilizando la función `DATEDIFF`

```
1 SELECT DATEDIFF('2021-01-31', '2021-02-01');
2 -> -1
```

Puede encontrar más información sobre la [conversión de tipos en la evaluación de expresiones en la documentación oficial](#).