

SQL (III)

DML

Consultas. SELECT

Introducción

El comando de SQL que se utiliza para consulta de datos es **SELECT**. 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 comando SELECT permite:

- Obtener datos de ciertas columnas de una tabla (**proyección**).
- Obtener registros (filas) de una tabla de acuerdo con ciertos criterios (**selección**).
- Mezclar datos de tablas diferentes (**asociación, join**)
- Realizar cálculos sobre los datos.
- Agrupar datos.

SINTAXIS SENCILLA DEL COMANDO SELECT

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

Una sintaxis completa del comando select puede consultarse en [la documentación oficial de MySQL](#)

SENTENCIA SELECT:

ORDEN DE EJECUCIÓN DE LAS CLÁUSULAS

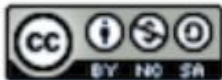
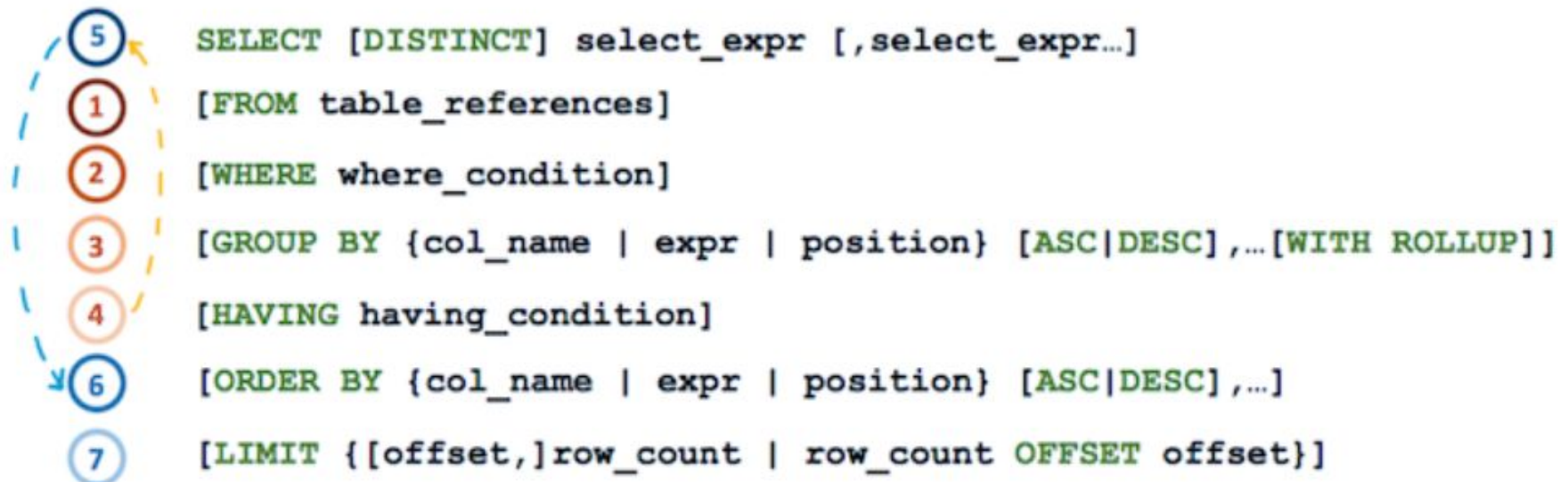
Es muy importante conocer en qué orden se ejecuta cada una de las cláusulas que forman la sentencia SELECT.

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).

SENTENCIA SELECT:

ORDEN DE EJECUCIÓN DE LAS CLÁUSULAS



COMENTARIOS EN CONSULTAS SQL

Hay diferentes formas de escribir comentarios en una consulta SQL.

```
-- Esto es un comentario
SELECT NOMBRE,APELLIDO1,APELLIDO2 -- esto es otro comentario
FROM ALUMNO
```

```
/* Esto es un comentario
de varias líneas */
SELECT NOMBRE,APELLIDO1,APELLIDO2 /* Esto es otro comentario*/
FROM ALUMNO;
```

```
# Esto es un comentario
SELECT NOMBRE,APELLIDO1,APELLIDO2 # Esto es otro comentario
FROM ALUMNO;
```

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:

- *, que sustituye a todas las columnas de la tabla o tablas que aparecen en la cláusula FROM.
- El nombre de una columna de la tabla sobre la que estamos realizando la consulta (una columna de la tabla que aparece en la cláusula FROM).
- Una expresión que nos permite calcular nuevos valores.
- Una constante que aparecerá en todas las filas de la tabla resultado.

TABLA PARA EJEMPLOS DEL TEMA

Para realizar los ejemplos, vamos a crear esta tabla:

```
CREATE TABLE ALUMNO (  
  ID INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
  NOMBRE VARCHAR(100) NOT NULL,  
  APELLIDO1 VARCHAR(100) NOT NULL,  
  APELLIDO2 VARCHAR(100),  
  FECHA_NACIMIENTO DATE NOT NULL,  
  ES_REPETIDOR ENUM('sí', 'no') NOT NULL,  
  TELEFONO VARCHAR(9)  
);
```

Suponemos que la tabla anterior tiene los siguientes datos almacenados:

```
INSERT INTO ALUMNO VALUES (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),  
                             (9, 'Manuel', 'Domínguez', 'Hernández', '1999/07/08', 'no', NULL),  
                             (10, 'Daniel', 'Moreno', 'Ruiz', '1998/02/03', 'no', NULL);
```


1. CONSULTAR O SELECCIONAR DATOS DE UNA TABLA

1.1 Selección de TODO los datos de todos los campos o columnas de una tabla.

SELECT * FROM ALUMNO;

El carácter * es un comodín que utilizamos para indicar que queremos seleccionar todos los campos o columnas de una tabla.

ID	NOMBRE	APELLIDO1	APELLIDO2	FECHA_NACIMIENTO	ES_REPETIDOR	TELEFONO
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
9	Manuel	Domínguez	Hernández	1999-07-08	no	NULL
10	Daniel	Moreno	Ruiz	1998-02-03	no	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

1. CONSULTAR O SELECCIONAR DATOS DE UNA TABLA

1.2. Selección de ALGUNOS campos o columnas de una tabla.

SELECT NOMBRE, APELLIDO1, APELLIDO2 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

Hay que tener en cuenta que el resultado de la consulta mostrará sólo las columnas que se han solicitado y en el orden en que se hayan indicado en el select.

ALIAS DE COLUMNAS (AS)

Si el nombre de la columna resulta demasiado largo, corto o es poco significativo, se utilizan alias de columnas para renombrarla, utilizando la palabra AS.

Ejemplo:

```
SELECT NOMBRE AS NOMBRE_ALUMNO  
FROM ALUMNO;
```

NOMBRE_ALUMNO
María
Juan
Pepe
Lucía

Si el alias de la columna se forma con más de una palabra y contiene espacios en blanco, el alias debe ir escrito entre comillas.

```
SELECT NOMBRE AS 'NOMBRE ALUMNO'  
FROM ALUMNO;
```

NOMBRE ALUMNO
María
Juan
Pepe
Lucía

SELECT CON CONDICIONES - CLÁUSULA WHERE

La cláusula WHERE nos permite añadir filtros a las consultas para seleccionar sólo aquellas filas que cumplen determinadas condiciones.

Estas condiciones se denominan predicados y el resultado de estas condiciones puede ser verdadero, falso o desconocido.

Ejemplo de consulta con cláusula WHERE:

```
SELECT * FROM ALUMNO  
WHERE ES_REPETIDOR='no' AND APELLIDO1='SÁNCHEZ'  
and FECHA_NACIMIENTO >'1990-01-01';
```

SELECT CON CONDICIONES - CLÁUSULA WHERE

Podemos diferenciar 5 tipos de condiciones que pueden aparecer en la cláusula WHERE.

- Condiciones para comprobar 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 de caracteres con un patrón.
- Condiciones para comprobar si una columna tiene valores a NULL.

CLÁUSULA WHERE. OPERANDOS Y OPERADORES

Los operandos usados en las condiciones pueden ser nombres de columnas, constantes, expresiones y los operandos pueden ser aritméticos, lógicos, de comparación, etc.

Pueden utilizarse tanto para comparar números como para comparar textos y fechas.

En el caso de textos se comparan en orden alfabético estricto, tomando el orden de tabla de códigos de caracteres.

CLÁUSULA WHERE. OPERADORES ARITMÉTICOS

OPERADOR	SIGNIFICADO
+	SUMA
-	RESTA
*	MULTIPLICACIÓN
/	DIVISIÓN
%	MÓDULO

CLÁUSULA WHERE. OPERADORES DE COMPARACIÓN

Pueden utilizarse tanto para comparar números como para comparar textos y fechas.

En el caso de textos se comparan en orden alfabético estricto, tomando el orden de tabla de códigos de caracteres.

OPERADOR	SIGNIFICADO
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
=	Igual
<> , !=	Distinto

CLÁUSULA WHERE.

OPERADORES LÓGICOS

OPERADOR	SIGNIFICADO
AND &&	Devuelve verdadero si las expresiones a su izquierda y derecha son ambas verdaderas.
OR 	Devuelve verdadero si cualquiera de las dos expresiones a izquierda y derecha del OR son verdaderas.
NOT !	Invierte la lógica de la expresión que está a su derecha, si era verdadera se convierte en falsa.

PRECEDENCIA DE OPERADORES

A veces las expresiones que aparecen en las consultas son muy complejas y debemos tener en cuenta qué parte de la expresión se evalúa primero.

Para ello, tendremos en cuenta el orden de precedencia de operadores (estas preferencias se siguen en los SGBD más usuales, no obstante, podría haber alguna variación en algún SGBD en concreto)

TABLA DE PRECEDENCIA DE OPERADORES

ORDEN DE PRECEDENCIA EN LA EJECUCIÓN	OPERADOR
1	*(Multiplicación) y / (División)
2	+ (suma) y – (resta)
3	(Concatenación)
4	Comparaciones (<,>!=,...)
5	IS NULL, IS NOT NULL, LIKE, NOT LIKE,IN
6	NOT
7	AND
8	OR

La precedencia de operadores puede alterarse si utilizamos paréntesis.

```
97 • SELECT APELLIDO1,APELLIDO2,NOMBRE,ES_REPETIDOR FROM ALUMNO
98 WHERE (NOMBRE='Manuel' OR NOMBRE='María') AND ES_REPETIDOR='NO';
```

<

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	APELLIDO1	APELLIDO2	NOMBRE	ES_REPETIDOR
▶	Sánchez	Pérez	María	no
	Domínguez	Hernández	Manuel	no

En este ejemplo, al resultado de la operación OR se le aplica el operador AND

BETWEEN

Este operador nos permite obtener datos cuyo valor esté dentro de un rango de valores

Sintaxis:

expresión [NOT] BETWEEN valor_inferior AND valor_superior

```
SELECT * FROM ALUMNO  
WHERE FECHA_NACIMIENTO BETWEEN '1990-01-01' AND '1995-12-31';
```

Devuelve los datos de los alumnos nacidos entre los años 1990 y 1995 (ambos incluidos)

ID	NOMBRE	APELLIDO1	APELLIDO2	FECHA_NACIMIENTO	ES_REPETIDOR	TELEFONO
1	María	Sánchez	Pérez	1990-12-01	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
8	Antonio	Carretero	Ortega	1994-05-20	sí	612345633

IN - NOT IN

Permite comprobar si el valor de una determinada columna está incluido en una lista o conjunto de valores.

Ejemplos:

```
SELECT NOMBRE, APELLIDO1, APELLIDO2
FROM ALUMNO
WHERE NOMBRE IN ('María', 'Paco', 'Antonio');
```

NOMBRE	APELLIDO1	APELLIDO2
María	Sánchez	Pérez
Paco	Martínez	López
Antonio	Carretero	Ortega

```
SELECT APELLIDO1, APELLIDO2, NOMBRE, ES_REPETIDOR
FROM ALUMNO
WHERE APELLIDO1 NOT IN ('Sánchez', 'Fernández');
```

APELLIDO1	APELLIDO2	NOMBRE	ES_REPETIDOR
Sáez	Vega	Juan	no
Ramírez	Gea	Pepe	no
Martínez	López	Paco	no
Gutiérrez	Sánchez	Irene	sí
Carretero	Ortega	Antonio	sí
Domínguez	Hernández	Manuel	no
Moreno	Ruiz	Daniel	no

LIKE - NOT LIKE

Sólo se aplica en campos de tipo texto (cadenas de caracteres).

Se utiliza para escribir consultas donde el criterio de selección en la cláusula WHERE es una parte de la cadena de caracteres (un patrón) donde se incluyen símbolos que funcionan como comodines sustituyendo los valores reales de los caracteres del campo consultado. Estos símbolos son el % y el _

SÍMBOLO	Sustituye a
%	Una cadena de caracteres cualquiera
_	Un carácter cualquiera

EJEMPLOS

```
SELECT NOMBRE, APELLIDO1, APELLIDO2
FROM ALUMNO
WHERE NOMBRE LIKE 'M%';
```

NOMBRE	APELLIDO1	APELLIDO2
María	Sánchez	Pérez
Manuel	Domínguez	Hernández

Obtiene los datos de los alumnos cuyo nombre empieza por M.

```
SELECT NOMBRE, APELLIDO1, APELLIDO2
FROM ALUMNO
WHERE NOMBRE LIKE '%N';
```

NOMBRE	APELLIDO1	APELLIDO2
Juan	Sáez	Vega

Obtiene los datos de los alumnos cuyo nombre termina por N.

```
SELECT NOMBRE, APELLIDO1, APELLIDO2
FROM ALUMNO
WHERE NOMBRE LIKE 'M%L';
```

NOMBRE	APELLIDO1	APELLIDO2
Manuel	Domínguez	Hernández

Obtiene los datos de los alumnos cuyo nombre empieza por M y termina por N.

```
SELECT NOMBRE, APELLIDO1, APELLIDO2
FROM ALUMNO
WHERE APELLIDO1 LIKE 'S_NCHEZ';
```

NOMBRE	APELLIDO1	APELLIDO2
María	Sánchez	Pérez
Lucía	Sánchez	Ortega

Obtiene los datos de los alumnos cuyo nombre contiene cualquier carácter en la segunda posición.

```
SELECT NOMBRE, APELLIDO1, APELLIDO2
FROM ALUMNO
WHERE APELLIDO1 LIKE '____';
```

NOMBRE	APELLIDO1	APELLIDO2
Juan	Sáez	Vega

Obtiene los datos de los alumnos cuyo nombre tenga 4 caracteres.

IS NULL - IS NOT NULL

IS NULL muestra los registros donde se cumple que el campo no contiene información.

IS NOT NULL muestra los registros donde se cumple que el campo contiene información

IS NULL - IS NOT NULL

Ejemplos:

```
SELECT NOMBRE, APELLIDO1  
FROM ALUMNO  
WHERE TELEFONO IS NULL;
```

NOMBRE	APELLIDO1
María	Sánchez
Pepe	Ramírez
Irene	Gutiérrez
Manuel	Domínguez
Daniel	Moreno

Selecciona los alumnos que NO tienen teléfono.

```
SELECT NOMBRE, APELLIDO1  
FROM ALUMNO  
WHERE TELEFONO IS NOT NULL;
```

NOMBRE	APELLIDO1
Juan	Sáez
Lucía	Sánchez
Paco	Martínez
Cristina	Fernández
Antonio	Carretero

Selecciona los alumnos que tienen algún valor en el campo teléfono.

CLÁUSULA ORDER BY

Esta cláusula permite ordenar las filas del resultado de la consulta en función del campo o campos que se indiquen.

La ordenación puede ser ascendente o descendente (por defecto se toma ascendente) y se permite ordenar por varias columnas estableciendo distintos niveles de ordenación.

La sintaxis es:

[ORDER BY {COLUMNA | EXPRESIÓN | POSICIÓN} [ASC | DES], ...]

ORDENACIÓN (CLÁUSULA ORDER BY)

Ejemplos:

```
SELECT * FROM ALUMNO  
ORDER BY APELLIDO1 DESC, NOMBRE DESC;
```

ID	NOMBRE	APELLIDO1	APELLIDO2	FECHA_NACIMIENTO	ES_REPETIDOR	TELEFONO
1	María	Sánchez	Pérez	1990-12-01	no	<small>NULL</small>
4	Lucía	Sánchez	Ortega	1993-06-13	sí	678516294
2	Juan	Sáez	Vega	1998-04-02	no	618253876
3	Pepe	Ramírez	Gea	1988-01-03	no	<small>NULL</small>
10	Daniel	Moreno	Ruiz	1998-02-03	no	<small>NULL</small>
5	Paco	Martínez	López	1995-11-24	no	692735409
6	Irene	Gutiérrez	Sánchez	1991-03-28	sí	<small>NULL</small>
7	Cristina	Fernández	Ramírez	1996-09-17	no	628349590
9	Manuel	Domínguez	Hernández	1999-07-08	no	<small>NULL</small>
8	Antonio	Carretero	Ortega	1994-05-20	sí	612345633

```
SELECT * FROM ALUMNO  
ORDER BY 3 DESC, NOMBRE DESC;
```

ID	NOMBRE	APELLIDO1	APELLIDO2	FECHA_NACIMIENTO	ES_REPETIDOR	TELEFONO
1	María	Sánchez	Pérez	1990-12-01	no	<small>NULL</small>
4	Lucía	Sánchez	Ortega	1993-06-13	sí	678516294
2	Juan	Sáez	Vega	1998-04-02	no	618253876
3	Pepe	Ramírez	Gea	1988-01-03	no	<small>NULL</small>
10	Daniel	Moreno	Ruiz	1998-02-03	no	<small>NULL</small>
5	Paco	Martínez	López	1995-11-24	no	692735409
6	Irene	Gutiérrez	Sánchez	1991-03-28	sí	<small>NULL</small>
7	Cristina	Fernández	Ramírez	1996-09-17	no	628349590
9	Manuel	Domínguez	Hernández	1999-07-08	no	<small>NULL</small>
8	Antonio	Carretero	Ortega	1994-05-20	sí	612345633

En las dos consultas se muestran los datos del alumno ordenados por APELLIDO1 descendente (el campo que está en la posición 3) y NOMBRE descendente.

CLÁUSULA LIMIT.

- La función de LIMIT es limitar el número de filas que se incluyen en el resultado de una consulta.
- También se usa para limitar el número máximo de registros a borrar en las consultas DELETE.
- Suele utilizarse después de una cláusula ORDER BY, para limitar la salida una vez que esta ha sido ordenada.

LIMITAR EL NÚMERO DE FILAS OBTENIDAS EN UNA CONSULTA. (CLÁUSULA LIMIT)

SINTAXIS:

[**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. La primera fila que se obtiene como resultado es la que sigue a la posición offset (offset+1)

EJEMPLO:



```
SELECT * FROM productos LIMIT 5,4;
```

Muestra 4 filas a partir de la fila nº 5

LIMITAR EL NÚMERO DE FILAS OBTENIDAS EN UNA CONSULTA. (CLÁUSULA LIMIT)



EJEMPLOS:

```
97 • SELECT *
98 FROM fabricantes
99 LIMIT 5;
```

<   Filter Rows:

	CODIGO	NOMBRE
▶	1	Asus
	2	Lenovo
	3	Hewlett-Packard
	4	Samsung
	5	Seagate
▲	NULL	NULL

```
• SELECT *
FROM fabricantes
LIMIT 3, 2;
```

ult Grid |   Filter Rows:

	CODIGO	NOMBRE
	4	Samsung
	5	Seagate
	NULL	NULL

UTILIZACIÓN DE FUNCIONES

- Todos los SGBD tienen funciones para facilitar la creación de consultas complejas.
- Dependen del SGBD que estemos utilizando.
- Devuelven un resultado de un determinado cálculo.
- La mayoría necesita que se le envíen datos de entrada (parámetros o argumentos), es decir, valores que necesita para hacer el cálculo.
- Las funciones se “invocan” (llaman) de la siguiente manera:
 - Nombre_Función [(parámetro1,parámetro2,.....)]

FUNCIONES DISPONIBLES EN MYSQL

- Las funciones que se muestran a continuación están disponibles en MySQL.
- Estas funciones pueden utilizarse en las consultas en las cláusulas SELECT, WHERE Y ORDER BY.
- [Enlace al manual de referencia de MySQL \(Funciones y operadores\)](#)

FUNCIONES MATEMÁTICAS

FUNCIÓN PARA REDONDEAR O TRUNCAR UN NÚMERO

Función	Descripción
ROUND (n,decimales)	Redondea el número al siguiente número con el número de decimales indicado más cercano. ROUND (8.239,2) devuelve 8.24
TRUNC (n,decimales)	Los decimales del número se cortan para que sólo aparezca el número de decimales indicado

EJEMPLOS:

SELECT ROUND('135.375', 2); DEVUELVE 135.38

SELECT TRUNC('135.375', 2); DEVUELVE 135.37

MÁS FUNCIONES MATEMÁTICAS

Función	Descripción
MOD($n1, n2$)	Devuelve el resto resultado de dividir $n1$ entre $n2$
POWER(valor,exponente)	Eleva el valor al exponente indicado
SQRT(n)	Calcula la raíz cuadrada de n
SIGN(n)	Devuelve 1 si n es positivo, cero si vale cero y -1 si es negativo
ABS(n)	Calcula el valor absoluto de n
EXP(n)	Calcula e^n , es decir el exponente en base e del número n
LN(n)	Logaritmo neperiano de n
LOG(n)	Logaritmo en base 10 de n
SIN(n)	Calcula el seno de n (n tiene que estar en radianes)
COS(n)	Calcula el coseno de n (n tiene que estar en radianes)
TAN(n)	Calcula la tangente de n (n tiene que estar en radianes)
ACOS(n)	Devuelve en radianes el arco coseno de n
ASIN(n)	Devuelve en radianes el arco seno de n
ATAN(n)	Devuelve en radianes el arco tangente de n
SINH(n)	Devuelve el seno hiperbólico de n
COSH(n)	Devuelve el coseno hiperbólico de n
TANH(n)	Devuelve la tangente hiperbólica de n

FUNCIONES CON CADENAS DE CARACTERES

CHAR_LENGTH(String): Retorna el número de caracteres de la cadena de texto. Ejemplo: `select char_length('Hola');` retorna un 4.

LENGTH(String): Retorna la longitud de la cadena de texto (medido en bytes). Ejemplo: `select char_length('Hola');` retorna un 4.

SUBSTRING(String [FROM INT] [FOR INT]): Retorna una parte de la cadena de caracteres, indicando la posición inicial y la cantidad de caracteres a extraer desde dicha posición. Ejemplo:

`select substring('Hola Mundo' from 1 for 2);` retorna 'Ho'.

UPPER(String): Retorna el texto convertido a mayúsculas. Ejemplo:

`select upper('Hola');` retorna 'HOLA'.

LOWER(String): Retorna el texto convertido a minúsculas. Ejemplo:

`select lower('Hola');` retorna 'hola'.

FUNCIONES CON CADENAS DE CARACTERES

CONCAT(string1,string2): Concatena dos o más cadenas de caracteres.

La función CONCAT() de MySQL no añade ningún espacio entre las columnas, por eso los valores de las dos columnas aparecen como una sola cadena sin espacios entre ellas.

Devuelve NULL cuando una de las cadenas es igual a NULL.

Ejemplo:

```
select concat ('Hola', 'Mundo'); retorna 'HolaMundo'.
```

```
select concat ('Hola','Mundo',NULL); Retorna NULL.
```

CONCAT_WS(separador,string1,string2): Concatena dos o más cadenas de caracteres e introduce una cadena como separador.

Omite todas las cadenas con valor NULL y concatena todas las demás.

Ejemplo:

```
select concat_ws ('---->','Hola','Mundo'); retorna Hola---->Mundo
```

```
select concat_ws ('---->','Hola','Mundo',NULL); retorna Hola---->Mundo
```

FUNCIONES CON CADENAS DE CARACTERES

REVERSE(string): Retorna la cadena str con el orden de los caracteres invertido. Ejemplo: `select reverse ('Hola');` retorna 'aloH'.

REPLACE(string, TEXTO, NUEVO TEXTO): reemplaza dentro de una cadena de caracteres, uno o varios caracteres por otros. Ejemplo:
`select replace ('Hola Mundo','o','*');` retorna 'H*la Mund*'.

OTRAS FUNCIONES CON CADENAS DE CARACTERES:

LEFT(): Devuelve los caracteres de una cadena, empezando por la izquierda

RIGHT(): Devuelve los caracteres de una cadena, empezando por la derecha

LTRIM(): Elimina los espacios en blanco del inicio de una cadena

RTRIM(): Elimina los espacios en blanco del final de una cadena

TRIM(): Elimina los espacios en blanco del inicio y del final de una cadena

[Todas las funciones con cadenas de caracteres](#)

FUNCIONES DE FECHA Y HORA en MYSQL

[NOW\(\)](#) : Devuelve la fecha y hora actual.

[CURTIME\(\)](#): Devuelve la hora actual

[YEAR\(FECHA\)](#): Devuelve el año de una fecha

[MONTH\(FECHA\)](#): Devuelve el mes de una fecha

[DAY\(FECHA\)](#): Devuelve el día de una fecha

[MONTHNAME\(FECHA\)](#): Devuelve el nombre del mes de una fecha.

[DAYNAME\(FECHA\)](#): Devuelve el nombre del día de una fecha.

[HOUR\(HORA\)](#): Devuelve las horas de un valor DATETIME.

[MINUTE\(HORA\)](#): Devuelve los minutos de una hora.

[DAYOFWEEK\(FECHA\)](#): retorna el índice del día de semana para la fecha pasada como parámetro.

Los valores de los índices son: 1=domingo, 2=lunes,... 7=sábado).

[DATE_FORMAT\(fecha,formato\)](#): Nos permite formatear fechas

[DATEDIFF\(fecha1,fecha2\)](#): Calcula el número de días que hay entre dos fechas

[Todas las funciones de FECHA y HORAS](#)

FUNCIONES DE FECHA en MYSQL

FUNCIONES PARA OBTENER LA FECHA Y HORA ACTUAL

CURRENT_DATE o **CURRENT_DATE()** Retorna la fecha actual

EJEMPLO:

<code>SELECT CURRENT_DATE;</code>	<code>CURRENT_DATE()</code>
<code>SELECT CURRENT_DATE();</code>	2022-01-22

CURRENT_TIME o **CURRENT_TIME()**: retorna la hora actual

EJEMPLO:

<code>SELECT CURRENT_TIME;</code>	<code>CURRENT_TIME()</code>
<code>SELECT CURRENT_TIME();</code>	17:41:06

CURRENT_TIMESTAMP retorna la fecha y la hora actual.

EJEMPLO:

<code>SELECT CURRENT_TIMESTAMP;</code>	<code>CURRENT_TIMESTAMP()</code>
<code>SELECT CURRENT_TIMESTAMP();</code>	2022-01-22 17:50:27

FUNCIÓN EXTRACT

EXTRACT (YEAR/MONTH/DAY FROM DATE): retorna una parte de la fecha según le indiquemos antes de FROM. Después de FROM irá un campo o valor de tipo fecha. Ejemplos:

```
SELECT EXTRACT(YEAR FROM '2022-1-25'); -- Devuelve el año 2022.  
SELECT EXTRACT(MONTH FROM '2022-1-25'); -- Devuelve el mes 1 (enero).  
SELECT EXTRACT(DAY FROM '2022-1-25'); -- Devuelve el día 25.
```

CALCULAR HORAS:

```
select extract(hour from '2022-1-25 12:25:50'); -- Retorna la hora '12'  
select extract(minute from '2022-1-25 12:25:50'); -- Retorna el minuto '25'  
select extract(second from '2022-1-25 12:25:50'); -- Retorna el segundo 50
```

CONSULTAS DE RESUMEN

USO DE FUNCIONES DE AGREGADO

CONSULTAS DE RESUMEN

- Una de las funcionalidades de la sentencia SELECT es permitir obtener resúmenes de los datos de las columnas de las tablas. Estas funciones realizan una operación específica sobre todas las filas de un grupo.
- Para ello la sentencia SELECT utiliza una serie de cláusulas específicas (GROUP BY, HAVING).
- La diferencia entre una consulta de resumen y una de las consultas anteriores es que en las consultas normales las filas del resultado se obtienen directamente de las filas del origen de datos y cada dato que aparece en el resultado tiene su dato correspondiente en el origen de la consulta mientras que las filas generadas por las consultas de resumen no representan datos del origen sino un total calculado sobre estos datos.

EJEMPLO

```
SELECT cod_Oficina, Region,  
       Ventas  
FROM Oficinas  
ORDER BY Region
```

```
SELECT Region, SUM (Ventas)  
FROM Oficinas  
GROUP BY Region
```

A la izquierda tenemos una consulta simple que muestra las oficinas con sus ventas ordenadas por región, y a la derecha una consulta de resumen que obtiene la suma de las ventas de las oficinas de cada región.

24	CENTRO	€60.30
23	CENTRO	
13	ESTE	€69.30
11	ESTE	€69.30
12	ESTE	€735.3
28	ESTE	€0,00
26	NORTE	
21	OESTE	€18.60
22	OESTE	€69.30

region character varying(20)	sum money
NORTE	
CENTRO	€60.30
OESTE	€87.90
ESTE	€873.9

Funciones de columna o Funciones de agregado

Una función de columna **se aplica a una columna** y obtiene un **valor que resume el contenido** de la misma. Las funciones de agregado más comunes son:

Función	Descripción
MAX(expr)	Calcula el valor máximo del grupo
MIN(expr)	Calcula el valor mínimo del grupo
AVG(exp)	Calcula el valor medio del grupo
SUM(expr)	Suma de todos los valores del grupo
COUNT (*)	Cuenta el número de filas que tiene como resultado de una consulta.
COUNT(columna)	Número de valores no nulos que hay en esa columna.

USO DE LA FUNCIÓN COUNT()

- COUNT(*) cuenta y devuelve el número de filas que retorna la consulta.
- COUNT(Columna) cuenta el número de filas en la cual la expresión no es null.
- COUNT(DISTINCT Columna). Cuenta los valores distintos de de la columna.
- Por defecto, la consulta se ejecuta contando todas las filas, incluyendo las repetidas.
- Si utilizamos SELECT COUNT(*), la función devuelve todas las filas, incluyendo duplicados y nulos.

EJEMPLOS DE USO DE LA FUNCIÓN COUNT

```
SELECT *  
FROM  
EJEMPLO_COUNT;
```

CAMPO1
NULL
NULL
1
1
2
2
2

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

COUNT(*)
7

Al poner *, cuenta todas las filas, incluye los nulos y repetidos.

```
SELECT COUNT(CAMPO1)  
FROM EJEMPLO_COUNT;
```

COUNT(CAMPO1)
5

Al poner un campo, cuenta todas las filas, sin incluir los nulos, pero sí los valores repetidos.

```
SELECT COUNT(DISTINCT(CAMPO1))  
FROM EJEMPLO_COUNT;
```

COUNT(DISTINCT(CAMPO1))
2

Ahora cuenta sólo los valores sin repetidos ni nulos.

```
SELECT COUNT(*) FROM  
(SELECT DISTINCT CAMPO1 FROM EJEMPLO_COUNT) AS CUENTA;  
Ahora cuenta también el valor NULL
```

COUNT(*)
3

AGRUPAMIENTO DE FILAS. GROUP BY

- La cláusula GROUP BY nos permite crear grupos de filas que tienen los mismos valores en las columnas por las que se desea agrupar.

```
SELECT AVG(SALARIO) AS 'SALARIO MEDIO', LOCALIDAD  
FROM PERSONAS  
GROUP BY(LOCALIDAD);
```

Calcula el salario medio de las personas que trabajan en cada localidad.

SALARIO MEDIO	LOCALIDAD
1767.050000	CACERES
1600.483333	MERIDA
2000.820000	BADAJOS

AGRUPAMIENTO DE FILAS. EJEMPLOS

Obtener el salario más alto de cada localidad.

```
SELECT MAX(SALARIO) AS 'SALARIO MAYOR', LOCALIDAD  
FROM PERSONAS  
GROUP BY(LOCALIDAD);
```

SALARIO MAYOR	LOCALIDAD
2200.55	CACERES
2000.25	MERIDA
2400.89	BADAJOS

CLÁUSULA HAVING

- La cláusula HAVING permite seleccionar sobre los grupos de filas que tienen los mismos valores en las columnas por las que queremos agrupar.
- HAVING también es la cláusula de selección cuando la condición utiliza una función de agregado.
- En una cláusula HAVING puede haber más de una condición combinadas con los operadores lógicos (and, or, not)
- La diferencia WHERE establece condiciones para la selección las filas que se obtienen de un SELECT mientras que HAVING establece condiciones para la selección de filas de un agrupamiento con GROUP BY.

EJEMPLOS

```
SELECT Region, COUNT(COD_OFICINA)
FROM Oficinas
GROUP BY Region
```

region character varying(20)	count bigint
NORTE	1
CENTRO	2
OESTE	2
ESTE	4

- Esta consulta devuelve el número de oficinas que hay en cada región .

```
SELECT Region, COUNT(COD_OFICINA)
FROM Oficinas
GROUP BY Region
HAVING REGION <>'CENTRO'
```

region character varying(20)	count bigint
NORTE	1
OESTE	2
ESTE	4

- Ahora la consulta devuelve el número de oficinas que hay en cada región, excepto cuando la región es centro. La consulta selecciona todos los registros, los agrupa para contarlos y luego elimina la fila donde la región es “centro”.

EJEMPLOS

```
SELECT COUNT(*) AS 'TOTAL PERSONAS', LOCALIDAD  
FROM PERSONAS  
GROUP BY(LOCALIDAD);
```

TOTAL PERSONAS	LOCALIDAD
3	CACERES
6	MERIDA
2	BADAJEZ

Esta consulta devuelve el número de personas que trabajan en cada localidad.

```
SELECT COUNT(*) AS 'TOTAL PERSONAS', LOCALIDAD  
FROM PERSONAS  
GROUP BY(LOCALIDAD)  
HAVING LOCALIDAD <> 'MERIDA';
```

TOTAL PERSONAS	LOCALIDAD
3	CACERES
6	MERIDA
2	BADAJEZ

Ahora la consulta devuelve el número de personas que hay en cada localidad, excepto para LOCALIDAD= MÉRIDA.

La consulta selecciona todas las filas, las agrupa por el valor del campo LOCALIDAD, las cuenta y filtra por la condición de la cláusula HAVING.

WHERE Y HAVING EN LA MISMA CONSULTA

```
SELECT AVG(SALARIO) AS 'SALARIO MEDIO', LOCALIDAD  
FROM PERSONAS  
WHERE FUNCION<>'DIRECTOR'  
GROUP BY(LOCALIDAD)  
HAVING LOCALIDAD <> 'MERIDA';
```

La consulta selecciona todos los registros que cumplan la condición que establece el WHERE, los agrupa para contarlos y luego elimina las filas que indica el HAVING

SALARIO MEDIO	LOCALIDAD
1767.050000	CACERES
1600.750000	BADAJOS

HAVING CON FUNCIONES DE AGREGADO

```
SELECT COUNT(*) AS 'TOTAL PERSONAS', LOCALIDAD  
FROM PERSONAS  
GROUP BY(LOCALIDAD)
```

TOTAL PERSONAS	LOCALIDAD
3	CACERES
6	MERIDA
2	BADAJOS

La consulta devuelve el número de personas que trabajan en cada localidad.

```
SELECT COUNT(*) AS 'TOTAL PERSONAS', LOCALIDAD  
FROM PERSONAS  
GROUP BY(LOCALIDAD)  
HAVING COUNT(*)>2;
```

TOTAL PERSONAS	LOCALIDAD
3	CACERES
6	MERIDA

Ahora la consulta devuelve el número de personas que trabajan en cada localidad, pero se eliminan las filas donde no se cumpla la condición de la cláusula HAVING.

HAVING CON FUNCIONES DE AGREGADO Y MÁS DE UNA CONDICIÓN

```
SELECT COUNT(*) AS 'TOTAL PERSONAS', LOCALIDAD  
FROM PERSONAS  
GROUP BY(LOCALIDAD)
```

TOTAL PERSONAS	LOCALIDAD
3	CACERES
6	MERIDA
2	BADAJOS

La consulta devuelve el número de personas que trabajan en cada localidad.

```
SELECT COUNT(*) AS 'TOTAL PERSONAS', LOCALIDAD  
FROM PERSONAS  
GROUP BY(LOCALIDAD)  
HAVING LOCALIDAD <> 'MERIDA' AND COUNT(*)>2;
```

TOTAL PERSONAS	LOCALIDAD
3	CACERES

Ahora la consulta devuelve el número de personas que trabajan en cada localidad, pero se eliminan las filas donde no se cumpla las dos condiciones de la cláusula HAVING.

HAVING CON FUNCIONES DE AGREGADO y ORDER BY

```
SELECT COUNT(*) AS 'TOTAL PERSONAS', LOCALIDAD  
FROM PERSONAS  
GROUP BY(LOCALIDAD)
```

TOTAL PERSONAS	LOCALIDAD
3	CACERES
6	MERIDA
2	BADAJOS

La consulta devuelve el número de personas que trabajan en cada localidad.

```
SELECT COUNT(*) AS 'TOTAL PERSONAS', LOCALIDAD  
FROM PERSONAS  
GROUP BY(LOCALIDAD)  
HAVING LOCALIDAD <> 'MERIDA'  
ORDER BY COUNT(*);
```

TOTAL PERSONAS	LOCALIDAD
2	BADAJOS
3	CACERES

Ahora la consulta devuelve el número de personas que trabajan en cada localidad, pero se eliminan las filas donde no se cumpla la condición de la cláusula HAVING y se ordena el resultado.

MÁS EJEMPLOS

Los campos que aparecen en una cláusula having tienen que estar presentes en una función de agrupamiento (group by) o en alguna función de agregado (min, max, avg).

```
SELECT COUNT(*) AS 'TOTAL PERSONAS', LOCALIDAD  
FROM PERSONAS  
GROUP BY(LOCALIDAD)  
HAVING LOCALIDAD <> 'MERIDA' AND FUNCION<>'DIRECTOR';
```

Error Code: 1054. Unknown column 'FUNCION' in 'having clause'

Esta consulta nos devuelve un error, porque el campo FUNCIÓN no aparece en el GROUP BY.