# *SQL.*

**TIPOS DE BBDD**

* RELACIONALES
* NO RELACIONALES

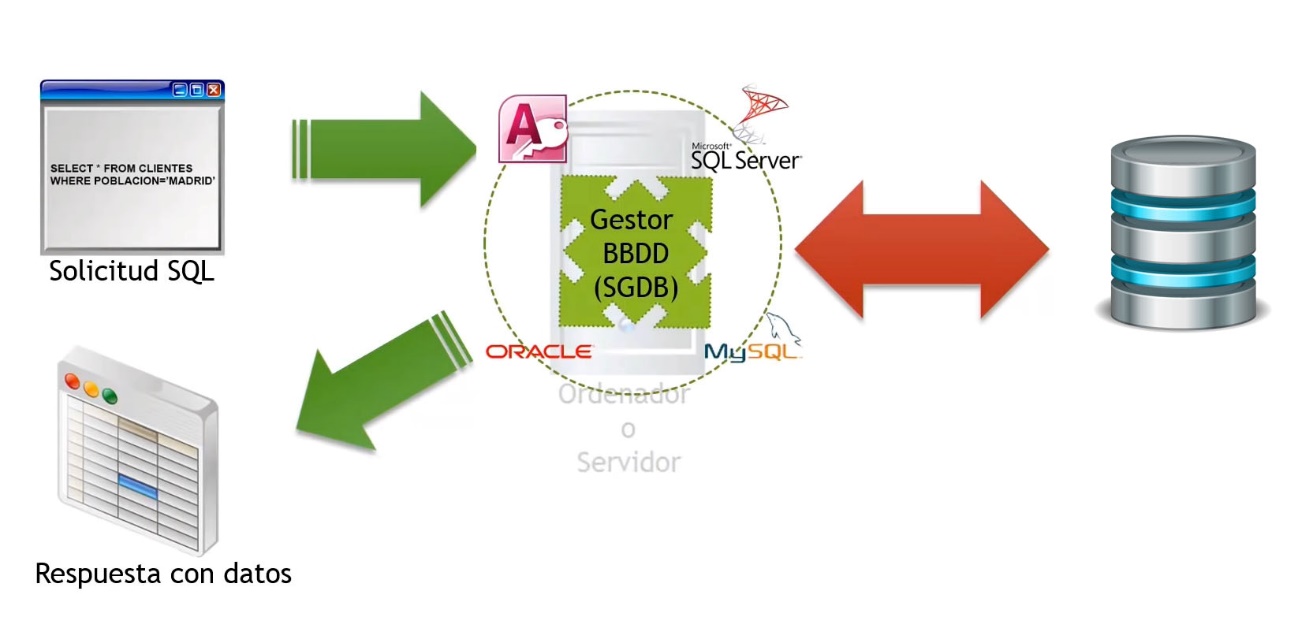
NoSQL (a veces llamado "no solo SQL") es una amplia clase de sistemas de gestión de **bases de datos** que difieren del modelo clásico de SGBDR (Sistema de Gestión de **Bases de Datos Relacionales**) en aspectos importantes, siendo el más destacado que no usan SQL como lenguaje principal de consultas.

**SQL** es un lenguaje estructurado creado por **IBM** para interactuar con bases de datos relacionales.

El alcance de **SQL** incluye la inserción de datos, consultas, actualizaciones y borrado, la creación y modificación de esquemas y el control de acceso los datos.

**SQL** es aceptado como el estándar para la creación y manipulación de **BBDD**. Después cada **SGBD** tendrá sus peculiaridades, pero en principio todos comparten los mismos principios.

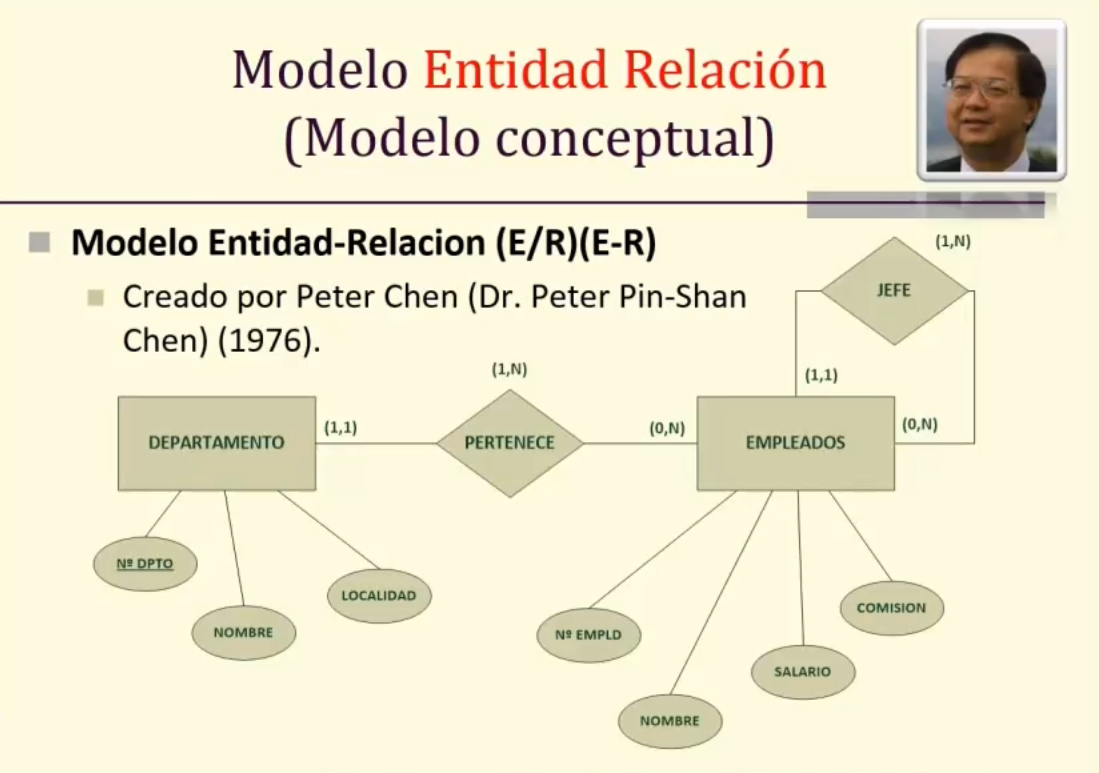
Nos encontraremos **SGBD** enfocados a trabajar en local, como “Acces” y otros enfocados a trabajar con servidores en red.



**Características de una BBDD relacional.**

* Una base de datos se compone de varias tablas, denominadas relaciones.
* No pueden existir dos tablas con el mismo nombre ni registro.
* Cada tabla es a su vez un conjunto de campos (columnas) y registros (filas).
* La relación entre una tabla padre y un hijo se lleva a cabo por medio de las llaves primarias y llaves foráneas (o ajenas).
* Las llaves primarias son la clave principal de un registro dentro de una tabla y estas deben cumplir con la integridad de datos.
* Las llaves ajenas se colocan en la tabla hija, contienen el mismo valor que la llave primaria del registro padre; por medio de estas se hacen las formas relacionales.

## Modelo entidad-relación.





## DISEÑO DE TABLAS

Convenciones a tener en cuenta:

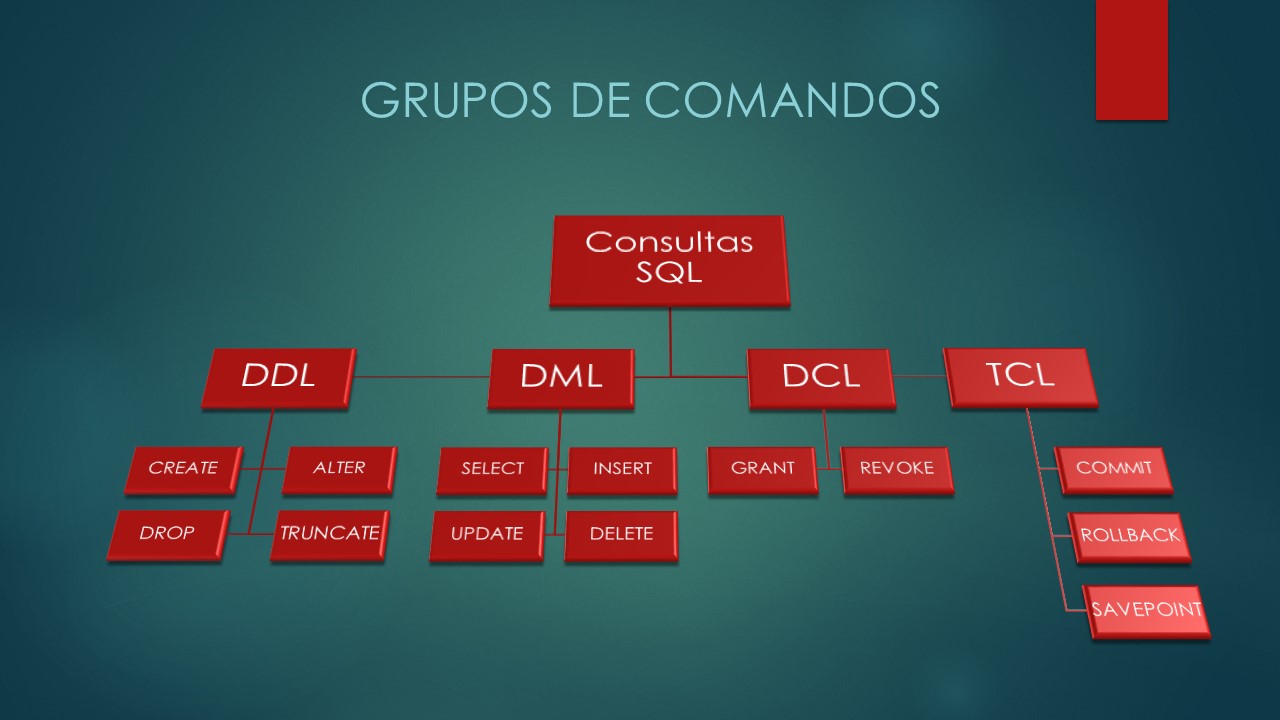
* Los comandos SQL se pondrán en mayúsculas.
* Todas las tablas tendrán una **primary** **key**.

# TIPOS DE DATOS.

|  |  |  |
| --- | --- | --- |
| **Tipos de datos númericos** | | |
| **Tipo** | **Definción** | **Bytes** |
| Integer, INT | Valores enteros con signo. | 4 |
| Numeric(n,m) | Números reales de hasta 18 digitos (con decimales), donde n representa el total de dígitos admitidos (normalmente denominado precisión) y m el número de posiciones decimales (escala). | 5-17 |
| Decimal(n,m) | Igual que el tipo numeric. | 5-17 |
| Float | Número de coma flotante, este tipo de datos se suele utilizar para los valores en notación cientifica. | 4-8 |
| **Tipos de datos alfanúmericos** | | |
| **Tipo** | **Definción** | **Bytes** |
| char(n) | Almacena de 1 a 255 caracteres alfanúmericos. Este valor viene dado por n, y es el tamaño utilizado en disco para almacenar dato. Es decir si defino un campo como char(255), el tamaño real del campo será de 255, aunque el valor solo contenga 100. | 0-255 |
| varchar(n) | Igual que el tipo char, con la salvedad que varchar almacena únicamente los bytes que contenga el valor del campo. | 0-255 |
| Nota:El tamaño del campo varÍa en función de cada base de datos, siendo 255 el valor standart. En realidad, el tamaño viene delimitado por el tamaño de las páginas de datos, para SQL Server el límite está en 8000 bytes (8000 caracteres), siempre y cuando tengamos definido el tamaño de la página de datos a 8K | | |
| **Tipos de datos fecha** | | |
| **Tipo** | **Definción** | **Bytes** |
| Date | Almacena fechas, con día, mes y año. | 8 |
| Datetime | Almacena fechas con fecha y hora | 4 |
| Nota:La aparición de los tipos de datos de fecha supuso una atentica revolución el mundo de la bases de datos, en realidad, la base de datos almacena internamente números enteros, de ahí que el tamaño sea de 4 bytes y 8 bytes (2 enteros), pero aporta la validación del dato introducido. | | |
| **Tipos de datos lógicos** | | |
| **Tipo** | **Definición** | **Bytes** |
| Bit | Tipo bit. Almacena un 0 ó no cero, según las bases de datos será 1 ó -1. Se aplica la lógica booleana, 0 es falso y no cero verdadero. | 1 bit |
| **Tipos de datos BLOB** | | |
| **Tipo** | **Definición** | **Bytes** |
| Image | Almacena imágenes en formato binario, hasta un máximo de 2 Gb de tamaño. | 0-2Gb |
| Text | Almacena texto en formato binario, hasta un máximo de 2 Gb de tamaño. | 0-2Gb |

* Lenguaje de definición de datos (DDL)
  + Create, Alter, Drop.
* Lenguaje de manipulación de datos (DML)
  + Insert, Update, Delete.
* Lenguaje de control de datos (DCL)
  + Grant, Revoke.
* Control de transacciones
  + Commit, Rollback, Savepoint
* Restricciones de integridad
  + Referencial, datos.

# GRUPOS DE COMANDOS.



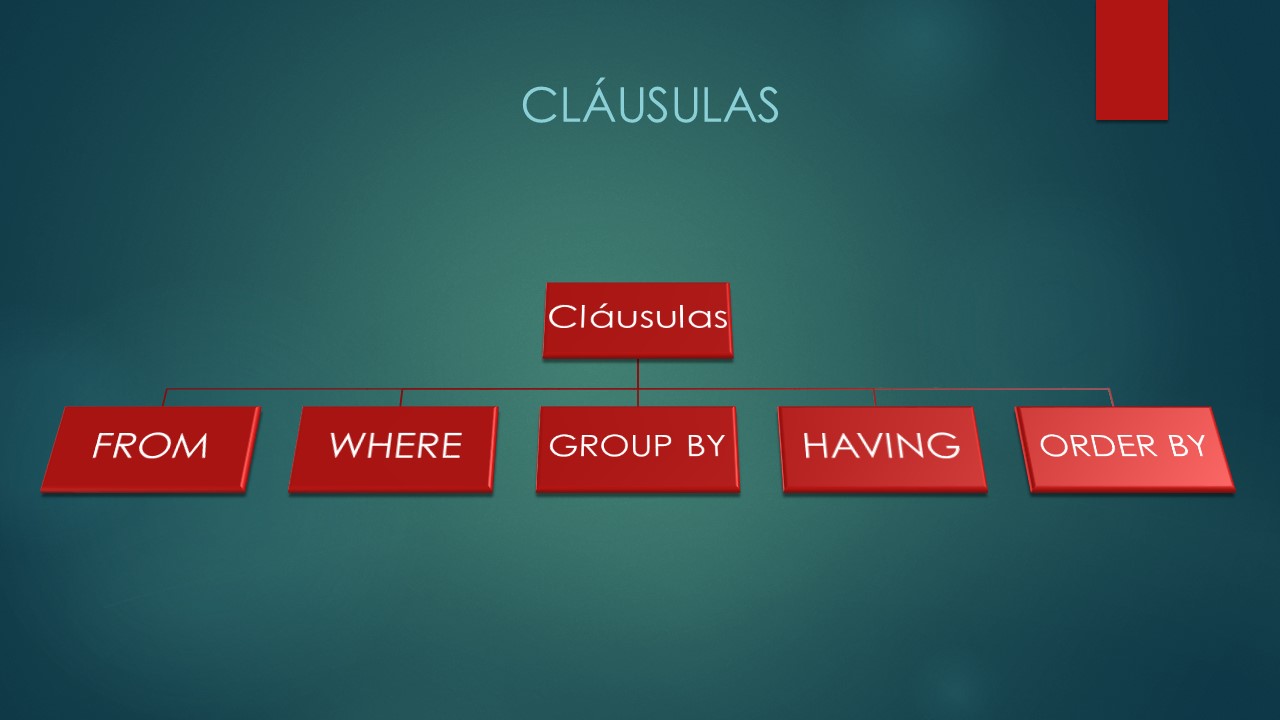
**DDL**: Data Definition Language. Comandos para crear y modificar la estructura de una **BBDD**.

**DML**: Data Manipulation Languaje. Consultas, de selección y de acción.

**DCL**: Data control lenguaje. Proporcionan seguridad a la información en la **BBDD**.

**TCL**: Transaction control lenguaje. Gestión de los cambios en los datos.

# CLÁUSULAS.



**FROM**: Especifica la tabla sobre la que vamos a trabajar.

**WHERE**: Especifica las condiciones o criterios de los registros seleccionados.

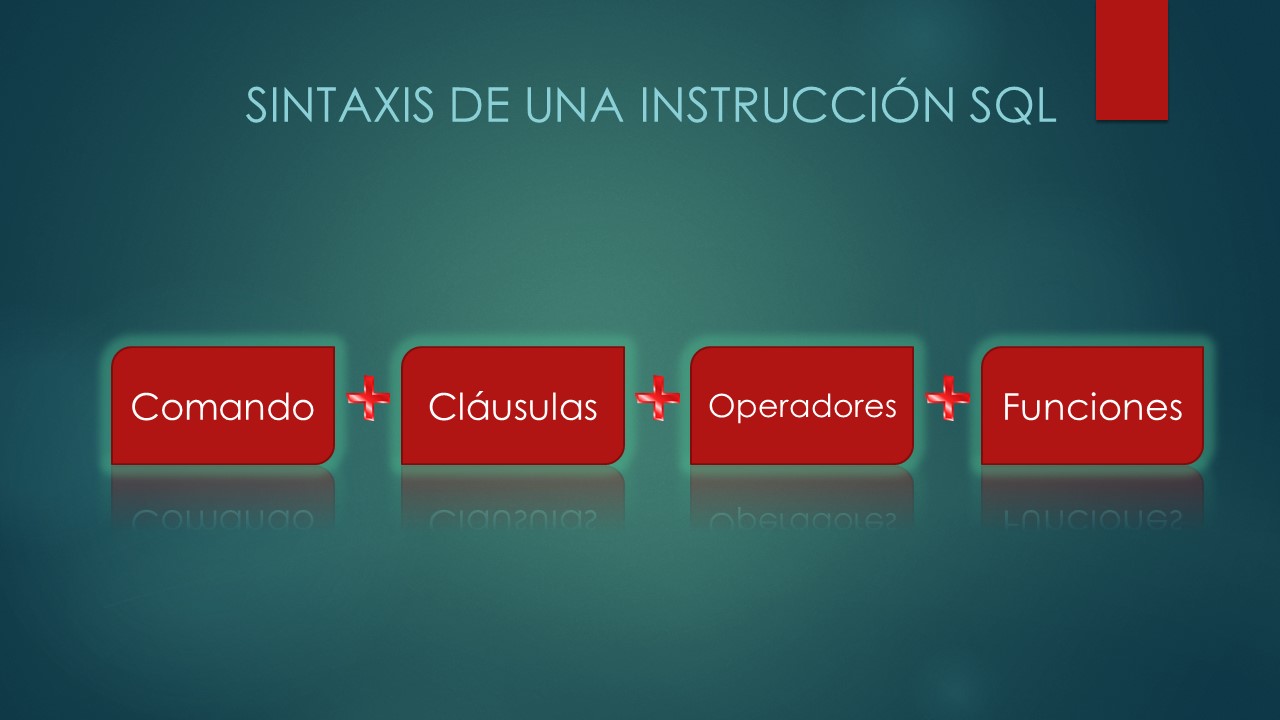
**GROUP BY**: Agrupa los registros seleccionados en función de un campo.

**HAVING**: Especifica las condiciones que deben cumplir los grupos.

**ORDER BY**: Ordena los registros seleccionados en función de un campo.

# OPERADORES DE COMPARACIÓN.

|  |  |
| --- | --- |
| **OPERADOR** | **SIGNIFICADO** |
| **<** | Menor que |
| **>** | Mayor que |
| **=** | Igual que |
| **<=** | Menor o igual que |
| **>=** | Mayor o igual que |
| **<>, ¡=** | Distinto que. |
| **BETWEEN** | Entre |
| **LIKE, NOT LIKE** | Para comparar caracteres. El wildcard es '%'. |
| **IN** | En. Especifica los registros en un campo concreto. |
| **AND** | “Y” lógico. |
| **OR** | “o” lógico. |
| **NOT** | Negación. |
| **IS NULL** | Es nulo |
| **IS NOT NULL** | No es nulo |





Ej:

SELECT \*

FROM clientes

**SELECT** EMPRESA, POBLACION

**FROM** clientes

**SELECT** NOMBRE, SECCION, PRECIO

**FROM** productos

**SELECT** NOMBRE, SECCION, PRECIO

**FROM** productos

**WHERE** SECCION="DEPORTES"

**SELECT** NOMBRE, SECCION, PRECIO

**FROM** productos

**WHERE** SECCION="HOGAR" **AND** SECCION="DEPORTES"🡪Incorrecto

**SELECT** NOMBRE, SECCION, PRECIO

**FROM** productos

**WHERE** SECCION="HOGAR" **OR** SECCION="DEPORTES" 🡪Correcto

**SELECT** \*

**FROM** CLIENTES

**WHERE** EMPRESA **LIKE** 'L%'

**SELECT** NOMBRE, SECCION, PRECIO

**FROM** productos

**WHERE** PRECIO<300

**SELECT** NOMBRE, PRECIO, FECHA

**FROM** productos

**WHERE** FECHA>='2001-01-01'

**SELECT** NOMBRE, PRECIO, FECHA

**FROM** productos

**WHERE** FECHA **BETWEEN** '2001-01-01' **AND** '2012-01-01'

**SELECT** \*

**FROM** `productos`

**WHERE** SECCION='DEPORTES'

**OR** SECCION='HOGAR'

**ORDER** BY SECCION **DESC**

Lo contrario de **DESC** sería **ASC**, pero no se suele usar ya que es la forma de ordenar por defecto.

Con la clausula **LIMIT** Podemos limitar el número de registros que me va a mostrar la query:

**SELECT** \*

**FROM** PRODUCTOS

**LIMIT** 3;

Me sacará solo 3 registros de la tabla.

Si le pongo otro parámetro…

**SELECT** \*

**FROM** PRODUCTOS

**LIMIT** 0,2;

…me sacará solo hasta el registro número 2.

También le puedo indicar el intervalo de registros que quiero que me saque:

**SELECT** \*

**FROM** PRODUCTOS

**LIMIT** 3,8;

**SELECT** \*

**FROM** `productos`

**WHERE** SECCION='HOGAR'

**OR** SECCION='DEPORTES'

**ORDER BY** SECCION,PRECIO

En las consultas también puedo meter operadores aritméticos:

[**select**](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/select.html) (7+7) **from** DUAL //Esto me daría 14

O por supuesto, operar con las filas de mi **BBDD**, por ejemplo sumarle un 10% al precio de mis artículos.

**SELECT** nombre, (PRECIO\*1.1)

**FROM** `productos` **AS** nuevo\_precio

# CONSULTAS DE AGRUPACION. GROUP BY.

También llamadas consultas totales. En estas consultas haremos una primera consulta (una agrupación) y sobre los resultados haremos una segunda consulta, es decir, necesitaremos por un lado el campo de agrupación y por otro al campo del cálculo.

Para ello se utilizan las funciones de agregado:

|  |  |
| --- | --- |
| **FUNCIÓN** | **SIGNIFICADO** |
| **AVG** | Calcula el promedio de un campo. |
| **COUNT** | Cuenta los registros de un campo. |
| **SUM** | Suma los registros de un campo. |
| **MAX** | Devuelve el máximo de un campo. |
| **MIN** | Devuelve el mínimo de un campo |

Ej:

Súmame el precio de todos los productos de cada sección:

**SELECT** SECCION, **SUM**(PRECIO)

**FROM** productos

**GROUP BY** SECCION

En estas consultas necesitamos un campo para agrupar y otro para operar.

Si quisiésemos ordenar los valores por precio no podríamos poner algo así:

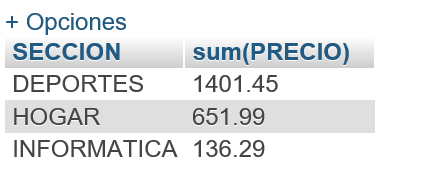
**SELECT** SECCION, sum(PRECIO)

**FROM** productos

**GROUP BY** SECCION

**ORDER BY** PRECIO

Ya que la variable precio no aparece en el resultado, en lugar de precio tenemos sum(precio):



Si para operar necesitamos trabajar sobre un resultado, le daremos un alias (con AS) y trabajaremos sobre él.

**SELECT** SECCION, sum(PRECIO)

**AS** sumaPrecios

**FROM** productos

**GROUP BY** SECCION

**ORDER BY** sumaPrecios+

En estas consultas, (con AVG, COUNT y SUM) la cláusula WHERE se sustituye por HAVING, con MAX y MIN se sigue usando WHERE.

**SELECT** SECCION, **AVG**(PRECIO) **AS** mediaPrecios

**FROM** productos

**GROUP BY** SECCION

**HAVING** SECCION='deportes'

**OR** SECCION='hogar'

En las consultas de agrupación, si incluimos un tercer campo en la búsqueda, hace lo que le da la gana.

**SELECT** SECCION, NOMBRE, AVG(PRECIO) **AS** mediaPrecios

**FROM** productos

**GROUP BY** SECCION

**HAVING** SECCION='deportes'

**OR** SECCION='hogar'

En este caso el nombre del artículo que nos muestra no es el de mayor precio, es el primero de la lista. Por eso **las consultas por agrupación deben tener sólo dos campos**.

**SELECT** POBLACION, **COUNT**(POBLACION) **AS** numClientesPorProvincia **FROM** clientes

**GROUP BY** POBLACION

La función COUNT no cuenta los campos vacíos, por eso se suele utilizar sobre campos clave.

Precio del artículo más caro de la sección de deportes.

**SELECT** SECCION, nombrearticulo, **MAX**(PRECIO)

**FROM** productos

**WHERE** SECCION='DEPORTES'

# CONSULTAS DE CÁLCULO.

También llamadas de campo calculado o de campo nuevo calculado.

Las consultas de cálculo se realizan sobre registros individuales, no sobre grupos como el caso de las consultas con **group by**.

Se suelen usar funciones, y cada gestor de **BBDD** suele tener las suyas propias, pero hay algunas comunes, como por ejemplo:

Funciones frecuentes:

-Now()

-Datediff()

-Date\_format()

-Concat()

**Now**(): Devuelve el día y la hora a la que se hace la consulta.

**Datediff**(fecha1,fecha2): Devuelve los días de diferencia que hay entre dos fechas.

**Date\_format**(fecha, '%D-%M') y format(): Nos permiten quitar decimales para redondear resultados. También se puede usar **ROUND**.

**Concat**(): Permite concatenar y se suele usar con cadenas de texto.

Esta función une dos datos dentro de una consulta, para que cuando los tengas que mostrar ya estén concatenados. Es muy útil por ejemplo cuando queremos mostrar direcciones o nombres y apellidos, pongamos un ejemplo:

**SELECT CONCAT**(Nombre, ' ', Apellidos) **AS** Nombre

**FROM** usuarios

Esta función nos extraerá el campo Nombre y Apellidos separados por un espacio, unidos en un solo campo llamado "**Nombre**", de esta manera tan sólo debemos mostrar un dato de información.

Esta función también se puede utilizar para concatenar valores numéricos:

**SELECT** NOMBRE, SECCION, **CONCAT**(PRECIO,'€')

**FROM** PRODUCTOS

Ej:

**SELECT** SECCION,PRECIO, PRECIO\*1.21 **AS** PRECIO\_CON\_IVA

**FROM** PRODUCTOS

**SELECT** SECCION,PRECIO, **ROUND**(PRECIO\*1.21,1) **AS** PRECIO\_CON\_IVA **FROM** PRODUCTOS

**SELECT** SECCION,PRECIO, FECHA,**NOW**() **AS** DIA\_DE\_HOY

**FROM** PRODUCTOS

**SELECT** NOMBRE,

SECCION,

PRECIO,

FECHA,

**DATE\_FORMAT**(**NOW**(), '%D-%M') **AS** DIA\_DE\_HOY,

**DATEDIFF (NOW()**, FECHA) **AS** DIAS\_DE\_DIFERENCIA

**FROM** PRODUCTOS

**WHERE** SECCION="DEPORTES"

# CONSULTAS MULTITABLA.

Cuando trabajamos con **BBDD**, con el paso del tiempo es muy frecuente que pasemos a tener información duplicada o dispersa en varias tablas.

Las consultas multitabla ( o consultas de unión), nos permiten recoger información de varias tablas simultáneamente. Con ellas consultamos varias tablas en una sola query.

En principio podemos hacer consultas multitabla simplemente poniendo las tablas que queremos consultar separadas por comas:

**SELECT** nombre

**FROM** clientes\_Malaga,

clientes\_Madrid;

E incluso podemos usar la nomenclatura del punto para especificar qué campos queremos mostrar de cada tabla:

**SELECT** clientes.nombre,

pedidos.poblacion

**FROM** clientes,

pedidos;

Se suele poner un alias para facilitar el trabajo y reducir código:

**SELECT** c.nombre,

p.poblacion

**FROM** clientes c,

pedidos p;

Este tipo de consultas es correcto pero poco eficaz, ya que lo que hace es recorrer todas las tablas recogiendo información y luego según la condición que le hayamos puesto va borrando lo que no interesa.

Ejemplos de consultas multitabla:

-Mostrar las entradas con el nombre del autor y el nombre de la categoría:

**SELECT** e.id ,

e.titulo,

u.nombre **AS** ‘Autor’,

c.nombre **AS** ‘Categoria’

**FROM** entradas e,

usuarios u,

categorías c

**WHERE** e.usuario\_id = u.id

**AND** e.categoria\_id = c.id;

-Mostrar el nombre de las categorías y al lado cuántas entradas tienen:

**SELECT** c.nombre,

**COUNT** (e.id)

**FROM** categorías c,

entradas e

**WHERE** e.categoria\_id = c.id

**GROUP BY** e.categoria\_id

-Mostrar el mail de los usuarios y al lado cuantas entradas tienen:

**SELECT** u.email,

**COUNT** (titulo)

**FROM** usuarios u,

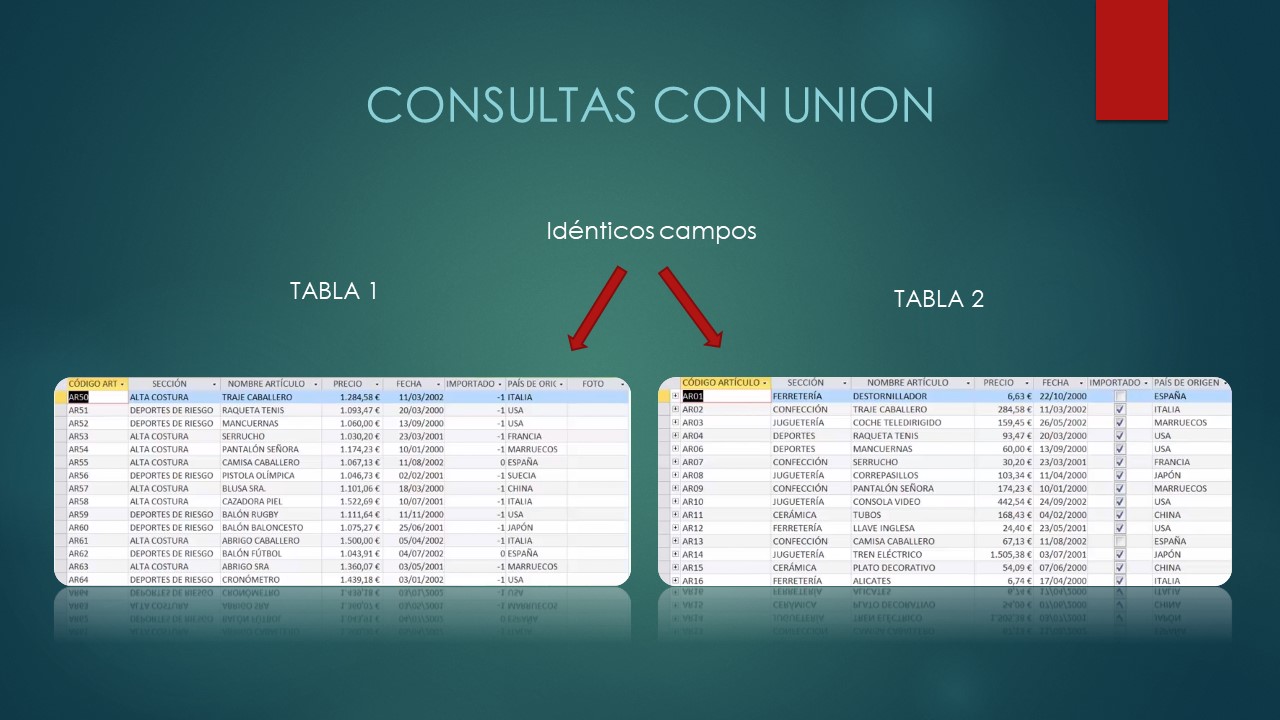
entradas e

**WHERE** e.usuario\_id = u.id

**GROUP BY** e.usuario\_id;

## CONSULTAS DE UNIÓN.

Para realizar las consultas multitabla, las tablas han de tener el **mismo número de campos** y estos deben ser **compatibles**. El nombre de los campos no tiene que ser idéntico.



Consultas multitabla / Consultas de unión:

* Unión externa:

-**Union**

-**Union all**

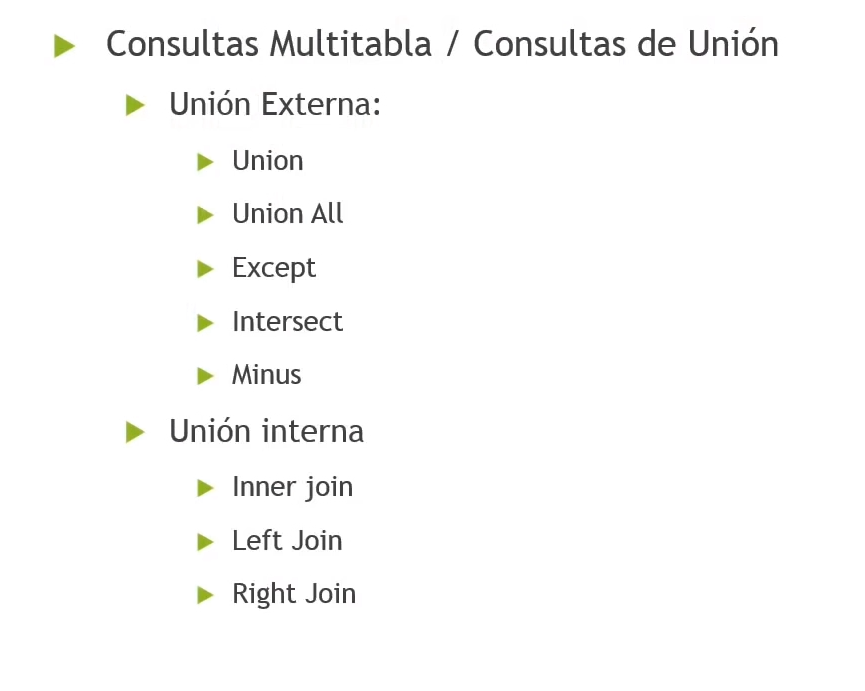
* Unión interna:

-**Inner join**

-**left join**

-**Right join**

## CONSULTAS MULTITABLA ERXTERNAS.



**UNION**: Muestra los campos seleccionados uniendo ambas tablas. Al campo resultante lo nombra como esté nombrado en la primera tabla.

Si hay un campo duplicado en ambas tablas lo muestra sólo una vez.

**UNION ALL**: Igual que UNION pero sí muestra los campos duplicados. Si está repetido 15 veces, 15 veces lo muestra.

**SELECT** \*

**FROM** PRODUCTOS

**WHERE** SECCION='DEPORTES'

**UNION**

**SELECT** \*

**FROM** PRODUCTOSNUEVOS

**WHERE** SECCION='DEPORTES DE RIESGO'

**SELECT** \*

**FROM** PRODUCTOS

**WHERE** PRECIO>500

**UNION**

**SELECT** \*

**FROM** PRODUCTOSNUEVOS

**WHERE** SECCION='ALTA COSTURA'

**SELECT** NOMBRE,

SECCION,

PRECIO

**FROM** PRODUCTOS

**WHERE** PRECIO>100

**UNION**

**SELECT** NOMBRE,

SECCION,

PRECIO

**FROM** PRODUCTOSNUEVOS

**WHERE** SECCION='DEPORTES DE RIESGO'

Como vemos al hacer este tipo de consultas tenemos que seleccionar los **mismos campos** en ambas tablas, en este caso hemos seleccionado los campos NOMBRE, SECCION y PRECIO de la primera tabla, luego en la segunda tengo que seleccionar **los mismos**.

Podemos concatenar tantos UNION como queramos.

Diferencia entre UNION y UNION ALL:

Con UNION los registros repetidos en ambas tablas los muestra sólo una vez.

UNION ALL muestra los registros repetidos tantas veces como aparezcan.

**SELECT** NOMBRE,

SECCION,

PRECIO

**FROM** PRODUCTOS

**WHERE** PRECIO>100

**UNION ALL**

**SELECT** NOMBRE,

SECCION,

PRECIO

**FROM** PRODUCTOSNUEVOS

**WHERE** SECCION='HOGAR'



## CONSULTAS MULTITABLA INTERNAS.



Un **INNER JOIN** nos devolvería la información común en ambas tablas. En este caso sólo los clientes que han hecho pedidos.

Un **LEFT JOIN** nos devuelve todos los registros de la tabla 1 y su relación con la tabla2. En este caso, **todos** los clientes y nos dice quienes han hecho pedidos. Es decir, mantiene todas las filas de la tabla 1 y luego me muestra también las coincidencias con la tabla 2, o NULL en caso de que no las hubiese.

Un **RIGHT JOIN** nos devuelve todos los registros de la tabla2 y su relación con la tabla1, es decir, **todos** los pedidos y los clientes que los han hecho.

Dicho de otra forma, mantiene todas las filas de la tabla 2 y luego me muestra también las coincidencias con la tabla 1, o NULL en caso de que no las hubiese.

A estas dos últimas, **LEFT JOIN** y **RIGHT JOIN** se les denomina también **OUTER JOINS**.

Ejemplo de **INNER** **JOIN**:

(Importante:Hay que decirle cuales son los campos que relacionan ambas tablas con ON)

Muéstrame los clientes de Madrid que han hecho pedidos:

**SELECT** \*

**FROM** CLIENTES

**INNER JOIN** PEDIDOS

**ON** clientes.CODIGO\_CLIENTE=pedidos.CODIGO\_CLIENTE

**WHERE** POBLACION='MADRID'

Ahora los que No han hecho pedidos:

**SELECT** \*

**FROM** CLIENTES

**LEFT JOIN** PEDIDOS

**ON** clientes.CODIGO\_CLIENTE=pedidos.CODIGO\_CLIENTE

**WHERE** POBLACION='MADRID'

**AND** pedidos.CODIGO\_CLIENTE **IS NULL**

**SELECT** CLIENTES.CODIGO\_CLIENTE,

EMPRESA,

POBLACION

**FROM** clientes

**INNER JOIN** pedidos

**ON** clientes.CODIGO\_CLIENTE=pedidos.CODIGO\_CLIENTE

**WHERE** POBLACION='MADRID'

**SELECT** CLIENTES.CODIGO\_CLIENTE,

EMPRESA,

POBLACION

**FROM** clientes

**LEFT JOIN** pedidos

**ON** clientes.CODIGO\_CLIENTE=pedidos.CODIGO\_CLIENTE

**WHERE** POBLACION='MADRID'

**AND** pedidos.CODIGO\_CLIENTE **IS NULL**

# VISTAS SQL

Son consultas almacenadas en la **BBDD** que se utiliza como una tabla virtual y no almacena datos, utiliza asociaciones y los datos de las tablas originales, de forma que siempre se mantiene actualizada.

**CREATE VIEW** mi\_Vista **AS**

**SELECT** u.email,

**COUNT** (titulo)

**FROM** usuarios u,

entradas e

**WHERE** e.usuario\_id = u.id

**GROUP BY** e.usuario\_id;

De este modo ya tenemos el resultado de la query almacenado en una vista llamada mi\_Vista y podremos usarlo repetidas veces, con lo que ahorramos código. Por lo que podríamos poner…

**SELECT** \*

**FROM** mi\_Vista;

…Y nos sacaría el resultado de la query.

Para borrar la vista:

**DROP VIEW** mi\_Vista;

# SUBCONSULTAS.

Las subconsultas no son más que una consulta dentro de otra, es decir, un **SELECT** dentro de otro **SELECT**. Así pues, tendremos un **SELECT** padre y dentro de él un **SELECT** hijo que irá entre paréntesis. Lo que hacemos es comparar lo que devuelve la consulta hija con lo que devuelve la consulta padre. Para ello utilizaremos los operadores lógicos y de comparación:

-Lógicos:

-**AND**: “y” lógico.

-**OR**: “o” lógico.

-**NOT**: Negación.

-De comparación:

-**LIKE**: Compara cadenas de caracteres.

SELECT \* FROM `clientes` WHERE EMPRESA **like** 'exportasa'

SELECT \* FROM `clientes` WHERE EMPRESA **like** 'e%'

**-<, >**, **=**, **<>**, **<=**, **>=** Menor, mayor, igual, distinto…

-**BETWEEN**: Rango entre intervalos, por ejemplo, entre fechas.

-**IN**: Están.

-**NOT IN**: No están.

-**ANY**: Cualquiera

-**ALL**: Todos.

Existen dos tipos de subconsultas:

**-Escalonadas**: Nos devuelven un campo con un registro, por ejemplo, la media de precios.

**-De lista**: Nos devuelven un campo y una lista de registros.

## -SUBCONSULTA ESCALONADA.

La consulta hija nos devuelven un campo con un registro, por ejemplo, la media de precios.

Nombre y sección de la tabla “productos” cuyo precio es mayor que la media:

**SELECT** NOMBRE,

SECCION,

PRECIO

**FROM** productos

**WHERE** PRECIO>(

**SELECT AVG**(PRECIO)

**FROM** productos)

Artículos cuyo precio es mayor que el más caro de deportes:

**SELECT** \*

**FROM** `productos`

**WHERE** precio>**ALL**(

**SELECT** PRECIO

**FROM** productos

**WHERE** SECCION='deportes')

Artículos cuyo precio es mayor que el más barato de deportes:

**SELECT** \*

**FROM** `productos`

**WHERE** precio>**ANY**(

**SELECT** PRECIO

**FROM** productos

**WHERE** SECCION='deportes')

Mostrar los artículos cuyo precio es superior a todos los precios de los artículos de HOGAR:

**SELECT** NOMBRE,

SECCION,

PRECIO

**FROM** productos

**WHERE** PRECIO<**ALL(**

**SELECT** PRECIO

**FROM** productos

**WHERE** SECCION='HOGAR'**)**

## -SUBCONAULTA DE LISTA.

Nos devuelven un campo y una lista de registros.

### PREDICADOS “**IN**” Y “**NOT IN**”.

Se puede traducir como “están” y “no están”. Se realiza una consulta en una tabla y se mira si están o no los resultados en la otra tabla.

Ej, nombre y precio de aquellos artículos de la tabla “productos” de los que se han pedido más de 20 unidades:

**SELECT** NOMBRE,

PRECIO

**FROM** PRODUCTOS

**WHERE** CODIGO\_ARTICULO **IN** (

**SELECT** CODIGO\_ARTICULO

**FROM** PRODUCTOS\_PEDIDOS

**WHERE** UNIDADES>20)

Mostrar datos de clientes que hayan realizado un pedido pagando con tarjeta:

**SELECT** CODIGO\_CLIENTE,

EMPRESA,

RESPONSABLE

**FROM** clientes

**WHERE** CODIGO\_CLIENTE **IN**(

**SELECT** CODIGO\_CLIENTE

**FROM** pedidos

**WHERE** FORMA\_PAGO='TARJETA')

Ahora los que no:

**SELECT** CODIGO\_CLIENTE,

EMPRESA,

RESPONSABLE,

FORMA\_PAGO

**FROM** clientes

**WHERE** CODIGO\_CLIENTE **NOT** **IN**(

**SELECT** CODIGO\_CLIENTE

**FROM** pedidos

**WHERE** FORMA\_PAGO='TARJETA')

Ejemplos de subconsultas:

- Sacar todos los usuarios que tengan alguna entrada que en su tÍtulo hable de GTA:

**SELECT** nombre,

apellidos

**FROM** usuarios

**WHERE** ID **IN**(

**SELECT** usuario\_id

**FROM** entradas

**WHERE** titulo **LIKE** %gta% );

-Sacar todas las entradas de la categoría acción utilizando su nombre:

**SELECT** titulo

**FROM** entradas

**WHERE** categoria\_id **IN**(

**SELECT** id

**FROM** categorias

**WHERE** nombre = ‘accion’);

-Mostrar las categorías con más de 3 entradas:

**SELECT** \*

**FROM** categorias

**WHERE** id **IN**(

**SELECT** categoria\_id

**FROM** entradas

**GROUP** **BY** categoria\_id

**HAVING** **COUNT** (categoria\_id>=3) );

-Mostrar los usuarios que crearon una entrada un martes:

**SEELECT** \*

**FROM** usuarios

**WHERE** id **IN**(

**SELECT** usuario\_id

**FROM** entradas

**WHERE** **DAYOFWEEK**(fecha)=3);

-Mostrar el nombre del usuario que tenga más entradas:

**SELECT** **CONCAT** (nombre, ‘ ‘, apellidos) **AS** ‘Usuario con mas entradas’ **FROM** usuarios

**WHERE** id =

(**SELECT** usuario\_id

**FROM** entradas

**GROUP BY** usuario\_id

**ORDER BY COUNT** (id) **DESC** **LIMIT** 1 );

-Mostrar las categorías sin entradas:

**SELECT** \*

**FROM** categorias

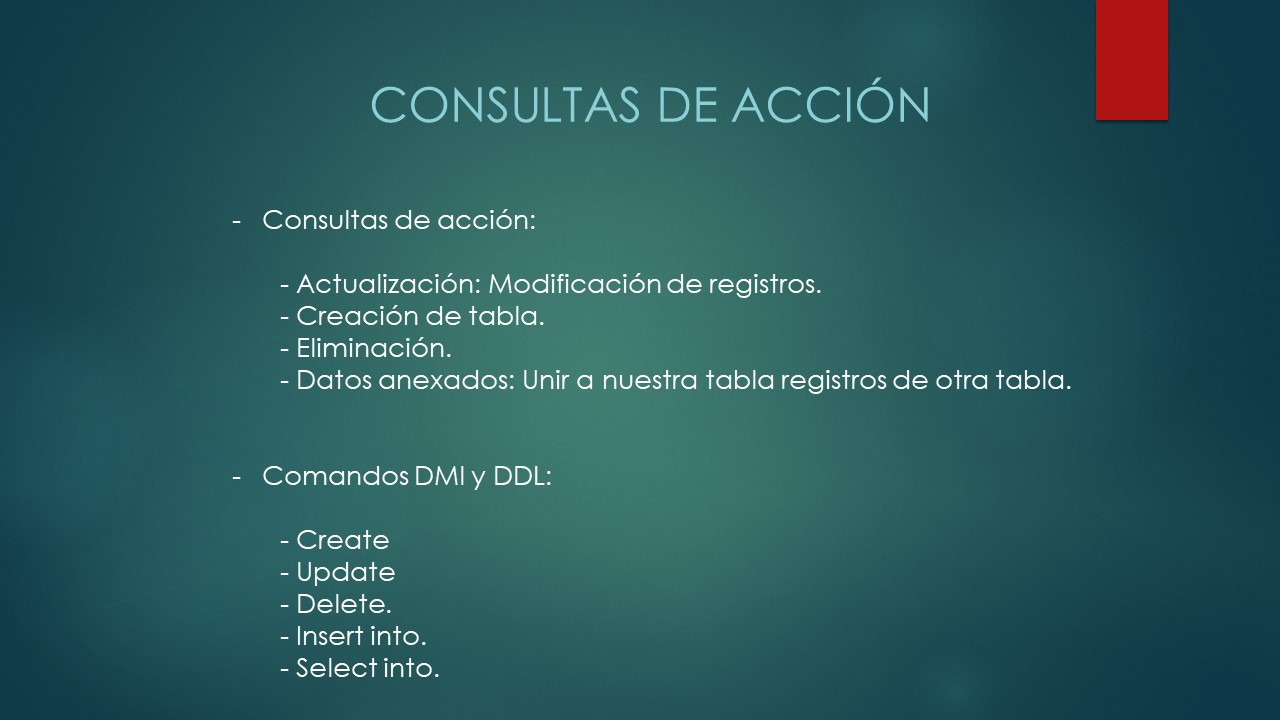
**WHERE** id **NOT** **IN**(

**SELECT** categoria\_id

**FROM** entradas);

# CONSULTAS DE ACCIÓN.

Modifican la información de las tablas.

****

**Las consultas de creación de tabla, se hacen de manera diferente según cada base de datos utilizada. OJO: No se pueden deshacer los cambios!!.**

Por lo que hay que estar muy atentos a poner un **WHERE** para que sólo se actualicen los campos que yo quiera, de lo contrario se actualizarían todos.

Ejemplo de consultas de actualización:

**UPDATE** PRODUCTOS

**SET** PRECIO=PRECIO+10

**WHERE** SECCION='DEPORTES'

Podemos hacer cambios en campos numéricos o cambios de tipo texto:

**UPDATE** PRODUCTOS

**SET** SECCION='DEPORTIVOS'

**WHERE** SECCION='DEPORTES' (Cambia el nombre del campo “deportes” por “deportivos”)

# CONSULTAS DE CREACIÓN.

Creamos una tabla a partir de los datos de otra. Se hacen de diferente forma según el gestor de **BBDD** que usemos.

**En ACCESS (SELECT INTO): SELECT \* INTO CLIENTES\_MADRID FROM CLIENTES WHERE POBLACION='MADRID'** (Crea una nueva tabla sólo con los clientes de Madrid)

**En MySQL (CREATE TABLE):**

**CREATE** **TABLE** CLIENTES\_MADRID

**SELECT** \*

**FROM** CLIENTES

**WHERE** POBLACION ='MADRID'

Ejemplos de consultas de eliminación:

**DELETE FROM** CLIENTES

**WHERE** POBLACION='MADRID'

**DELETE FROM** PRODUCTOS

**WHERE** SECCION='DEPORTES'

**AND** PRECIO **BETWEEN** 50 **AND** 100

**DELETE FROM** PRODUCTOS

**WHERE** SECCION='DEPORTES'

**AND** SECCION='HOGAR'

A la hora de trabajar con consultas DELETE hay que usar lo que se conoce como consultas de predicado, en las que utilizaremos un predicado como DISTINCT o DISTINCTROW

Clientes que han hecho pedidos:

**SELECT** EMPRESA

**FROM** clientes

**INNER JOIN** pedidos

**ON** clientes.CODIGO\_CLIENTE=pedidos.CODIGO\_CLIENTE

Clientes que han hecho pedidos sin repetirse ninguno:

**SELECT** DISTINCT EMPRESA

**FROM** CLIENTES

**INNER JOIN** PEDIDOS

**ON** CLIENTES.CODIGO\_CLIENTE=PEDIDOS.CODIGO\_CLIENTE

(“**Distinct**” me dice que no me repita el campo, en este caso la empresa, con que salga una sola vez me vale. Se pueden poner varios campos: distinct empresa, población……ni repite empresa ni población).

Con DISTINCT no me repite campos duplicados, DISTINCTROW no me repite registros duplicados.

Eliminación de clientes que no hayan hecho pedidos:

**DELETE** DISTINCTROW CLIENTES.\*,

PEDIDOS.CODIGO\_CLIENTE

**FROM** CLIENTES

**LEFT JOIN** PEDIDOS

**ON** CLIENTES.CODIGO\_CLIENTE=PEDIDOS.CODIGO\_CLIENTE

**WHERE** PEDIDOS.CODIGO\_CLIENTE **IS NULL**

# CONSULTAS DE DATOS ANEXADOS.

Nos permiten agregar registros de una tabla2 a una tabla1, los campos de ambas tablas han de ser idénticos.

**INSERT INTO** CLIENTES

**SELECT** \*

**FROM** CLIENTES\_MADRID

Si anexamos solo determinados campos tendremos que especificar dichos campos en ambas tablas:

**INSERT INTO** CLIENTES (

CODIGO\_CLIENTE,

EMPRESA,

DIRECCION)

**SELECT** CODIGO\_CLIENTE,

EMPRESA,

DIRECCION,

CODIGO\_CLIENTE

**FROM** CLIENTES\_MADRID

En estas consultas de indexación es obligatorio indexar los campos clave y los requeridos si existiesen, de lo contrario nos dará error. Además, como los elementos de los campos clave no se pueden repetir hay que asegurarse de que sean diferentes en ambas tablas.

**INSERT INTO** CLIENTES(

EMPRESA,

DIRECCION)

**SELECT** CODIGO\_CLIENTE,

EMPRESA,

DIRECCION

**FROM** CLIENTES\_MADRID

Falta CODIGO\_CLIENTE que es clave.

**INSERT INTO** CLIENTES(

NOMBRE,

POBLACION)

**VALUES** (‘Manolo’,

’Madrid’)

Al hacer un **insert** temenos que meter forzosamente los campos obligatorios de la tabla, en caso de que los tenga o nos dará error.

En este caso le hemos dicho que meta solo **nombre** y **población**, los demás campos, en caso de existir, me los dejaría **null**.

Si queremos hacer un **insert** rellenando todos los campos de la tabla, directamente ponemos los **values** sin especificar los campos que queremos rellenar. (se supone que todos):

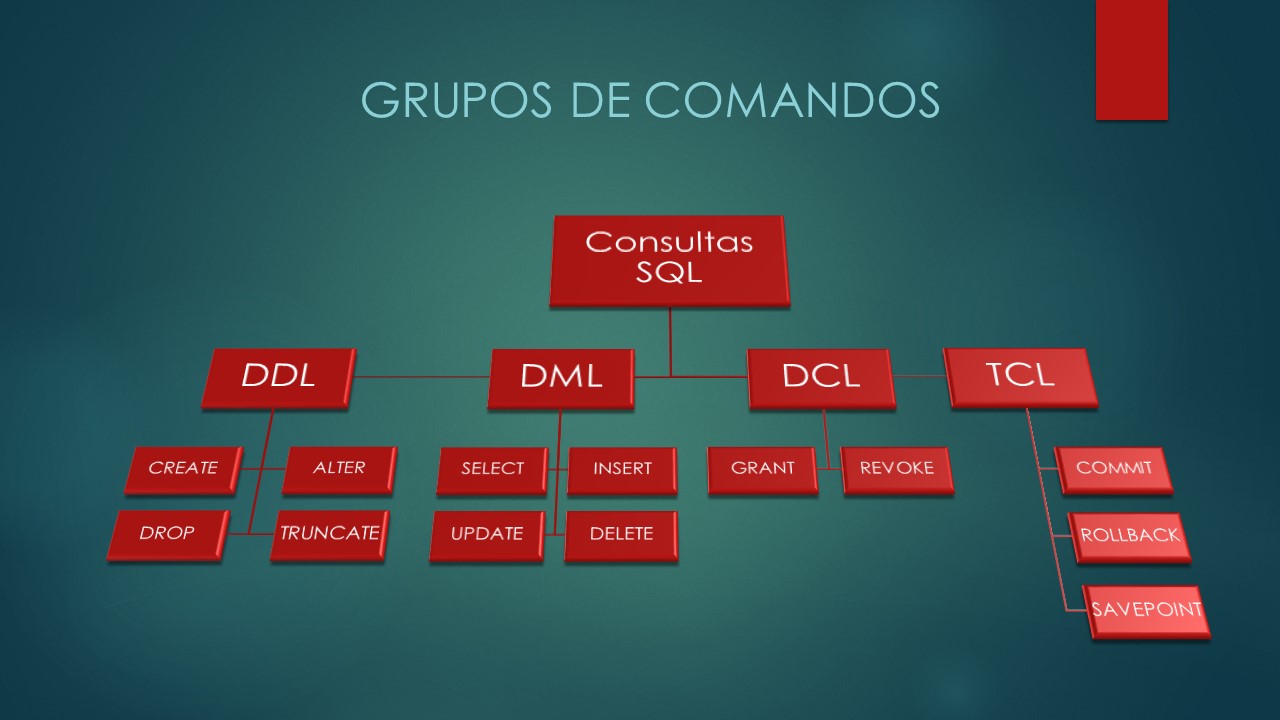
**INSERT INTO** CLIENTES

**VALUES**(‘Manolo’,

’Madrid’,

‘91987789’)

# CREACIÓN DE TABLAS.



**Comandos DDL:**

**- CREATE**: Crea nuevas **BBDD** y tablas.

**-ALTER**: Modifica los campos de las tablas.

**-DROP**: Elimina **BBDD** y tablas.

**-TRUNCATE**: Borra todas las filas de una tabla sin eliminar la tabla. Es decir, la deja a cero.

Sintaxis:

**CREATE TABLE** NOMBRE\_TABLA (

CAMPO1 TIPO\_DATO(LONGITUD),

CAMPO2 TIPO\_DATO(LONGITUD) ,

CAMPO3 TIPO\_DATO(LONGITUD),

…...)



Para crear una tabla primero tenemos que poner el comando **USE** seguido del nombre de la **BBDD**. En **PHP myAdmin** no hace falta, sólo tenemos que estar dentro de la **BBDD** en la que vamos a trabajar.

**USE** miBBDD;

Ej:

**CREATE TABLE** PRUEBA(

NOMBRE VARCHAR(20),

EDAD INT(2))

**DROP** TABLE PRUEBA

En MySQL:

**CREATE TABLE** PRUEBA(

NOMBRE VARCHAR(20),

EDAD INT(3),

FECHA\_NACIMIENTO DATE,

CASADO BOOLEAN)

Para añadir campos autoincrementados:

En Access:

**CREATE TABLE** PRUEBA(

ID\_ALUMNO COUNTER,

NOMBRE TEXT(20),

EDAD BYTE,

FECHA\_NACIMIENTO DATE,

CASADO BIT)

En MySQL:

CREATE TABLE PRUEBA(

ID\_ALUMNO INT AUTO\_INCREMENT,

NOMBRE TEXT(20),

EDAD INT(3),

FECHA\_NACIMIENTO DATE,

CASADO BOOLEAN,

PRIMARY KEY(ID\_ALUMNO))

EN **MySQL** los campos autoincrementados tienen que ser identificados como campo clave. Si voy a hacer un **insert** en una tabla con un campo autoincrementado, dicho campo lo pondré como **null**, ya se encarga **SQL** de darle el valor que corresponda.

Para añadir o quitar campos a una tabla ya creada:

Igual en Access y MySQL:

**ALTER TABLE** PRUEBA

**ADD** **COLUMN** FECHA\_BAJA DATE

**ALTER TABLE** PRUEBA

**DROP COLUMN** FECHA\_BAJA

**ALTER TABLE** PRUEBA

**ADD COLUMN** POBLACION VARCHAR(20)

Para cambiar las propiedades de un campo, usaremos el comando **ALTER** dos veces:

**ALTER TABLE** PRUEBA2

**ALTER COLUMN** FECHA\_BAJA TEXT(20)

Establecer el valor por defecto de un campo dentro de una tabla, usaremos el comando SET\_DEFAULT, Access no lo admite:

**ALTER TABLE** prueba1

**ALTER COLUMN** LUGAR\_NACIMIENTO

**SET\_DEFAULT** 'DESCONOCIDO'

Para quitar el valor por defecto:

**ALTER TABLE** prueba1

**ALTER COLUMN** LUGAR\_NACIMIENTO

**DROP** **DEFAULT**

Cambiar el nombre de una tabla:

**ALTER TABLE** prueba1

**RENAME TO** prueba2;

Cambiar el nombre de una columna:

**ALTER TABLE** prueba1

**CHANGE** columna1 nueva\_columna1 varchar(20);

Hemos cambiado el nombre de la columna1 por nueva\_columna1, hay que indicar también el tipo de dato que va a incluir, en este caso varchar(20).

Modificar una columna sin cambiar el nombre:

**ALTER TABLE** prueba1

**MODIFY** columna1 varchar(30);

Para ver una tabla además de un SELECT \* FROM myTabla, podemos usar también **DESCRIBE** myTabla o **DESC** myTabla. Con ello obtendremos una descripción completa de la tabla.

Añadir columnas a una tabla:

**ALTER TABLE** miTabla

**ADD** nueva\_columna varchar(20);

Para añadir restricción a una columna:

**ALTER TABLE** usuarios

**ADD CONSTRAINT** uq\_email **UNIQUE**(email);

De esta forma le hemos dicho a la **BBDD** que no puede haber un usuario con el mismo mail que otro.

# RESTRICCIONES DE INTEGRIDAD.

La restricción más común es el not null, que nos obliga a introducir siempre un dato en ese registro, no puede quedar vacío. Por el contrario, null permite lo contrario, que el valor sí pueda ser nulo.

Otra restricción es el default que otorga un valor por defecto.

También podemos poner una restricción de campo autoincrementado con auto\_increment. Dicha restricción sólo se puede usar en una columna que tenga clave primaria. **Recordemos que todas las tablas obligatoriamente tienen que tener una clave primaria**.

La restricción unique evita que se repitan campos. Por ejemplo, para evitar que haya dos usuarios con un mismo email.

**CREATE TABLE** miTabla(

Id number(255) **auto\_incremen**t,

nombre varchar(20) **not null**,

apellido varchar(20) **default** ‘Desconocido’)

# APÉNDICES DE GRUPOS Y SUB-GRUPOS.

Los tipos de datos que puede haber en un campo, se pueden agrupar en tres grandes grupos:

* [Tipos numéricos](https://www.desarrolloweb.com/articulos/1054.php#numericos)
* [Tipos de Fecha](https://www.desarrolloweb.com/articulos/1054.php#fecha)
* [Tipos de Cadena](https://www.desarrolloweb.com/articulos/1054.php#cadena)
* Tipo boolean

1- **Tipos numéricos**:

Existen tipos de datos numéricos, que se pueden dividir en dos grandes grupos, los que están en coma flotante (con decimales) y los que no.

**TinyInt**:

Es un número entero con o sin signo. Con signo el rango de valores válidos va desde -128 a 127. Sin signo, el rango de valores es de 0 a 255

**Bit** ó **Bool**:

Un número entero que puede ser 0 ó 1

**SmallInt**:

Número entero con o sin signo. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535.

**MediumInt**:

Número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607. Sin signo el rango va desde 0 a16777215.

**Integer**, **Int**:

Número entero con o sin signo. Con signo el rango de valores va desde -2147483648 a 2147483647. Sin signo el rango va desde 0 a 429.4967.295

**BigInt**:

Número entero con o sin signo. Con signo el rango de valores va desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.

**Float**:

Número pequeño en coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38.

**xReal**, **Double**:

Número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308

**Decimal**, **Dec**, **Numeric**:

Número en coma flotante desempaquetado. El número se almacena como una cadena

|  |  |
| --- | --- |
| **Tipo de Campo** | **Tamaño de Almacenamiento** |
| TINYINT | 1 byte |
| SMALLINT | 2 bytes |
| MEDIUMINT | 3 bytes |
| INT | 4 bytes |
| INTEGER | 4 bytes |
| BIGINT | 8 bytes |
| FLOAT(X) | 4 ú 8 bytes |
| FLOAT | 4 bytes |
| DOUBLE | 8 bytes |
| DOUBLE PRECISION | 8 bytes |
| REAL | 8 bytes |
| DECIMAL(M,D | M+2 bytes sí D > 0, M+1 bytes sí D = 0 |
| NUMERIC(M,D) | M+2 bytes if D > 0, M+1 bytes if D = 0 |

2- **Tipos fecha**:

A la hora de almacenar fechas, hay que tener en cuenta que Mysql no comprueba de una manera estricta si una fecha es válida o no. Simplemente comprueba que el mes está comprendido entre 0 y 12 y que el día está comprendido entre 0 y 31.

**Date**:

Tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia

**DateTime**:

Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos

**TimeStamp**:

Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037. El formato de almacenamiento depende del tamaño del campo:

|  |  |
| --- | --- |
| **Tamaño** | **Formato** |
| 14 | AñoMesDiaHoraMinutoSegundo aaaammddhhmmss |
| 12 | AñoMesDiaHoraMinutoSegundo aammddhhmmss |
| 8 | ñoMesDia aaaammdd |
| 6 | AñoMesDia aammdd |
| 4 | AñoMes aamm |
| 2 | Año aa |

**Time**:

Almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS'

**Year**:

Almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155. El campo puede tener tamaño dos o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

|  |  |
| --- | --- |
| **Tipo de Campo** | **Tamaño de Almacenamiento** |
| DATE | 3 bytes |
| DATETIME | 8 bytes |
| TIMESTAMP | 4 bytes |
| TIME | 3 bytes |
| YEAR | 1 byte |

3- **Tipos de cadena**:

**Char**(n):

Almacena una cadena de longitud fija. La cadena podrá contener desde 0 a 255 caracteres.

**VarChar**(n):

Almacena una cadena de longitud variable. La cadena podrá contener desde 0 a 255 caracteres.

Dentro de los tipos de cadena se pueden distinguir otros dos subtipos, los tipo Test y los tipo BLOB (Binary large Object)

La diferencia entre un tipo y otro es el tratamiento que reciben a la hora de realizar ordenamientos y comparaciones. Mientras que el tipo test se ordena sin tener en cuenta las Mayúsculas y las minúsculas, el tipo BLOB se ordena teniéndolas en cuenta.

Los tipos BLOB se utilizan para almacenar datos binarios como pueden ser ficheros.

**TinyText** y **TinyBlob**:

Columna con una longitud máxima de 255 caracteres.

**Blob** y **Text**:

Un texto con un máximo de 65535 caracteres.

**MediumBlob** y **MediumText**:

Un texto con un máximo de 16.777.215 caracteres.

**LongBlob** y **LongText**:

Un texto con un máximo de caracteres 4.294.967.295. Hay que tener en cuenta que debido a los protocolos de comunicación los paquetes pueden tener un máximo de 16 Mb.

**Enum**:

Campo que puede tener un único valor de una lista que se especifica. El tipo Enum acepta hasta 65535 valores distintos

**Set**:

Un campo que puede contener ninguno, uno ó varios valores de una lista. La lista puede tener un máximo de 64 valores.

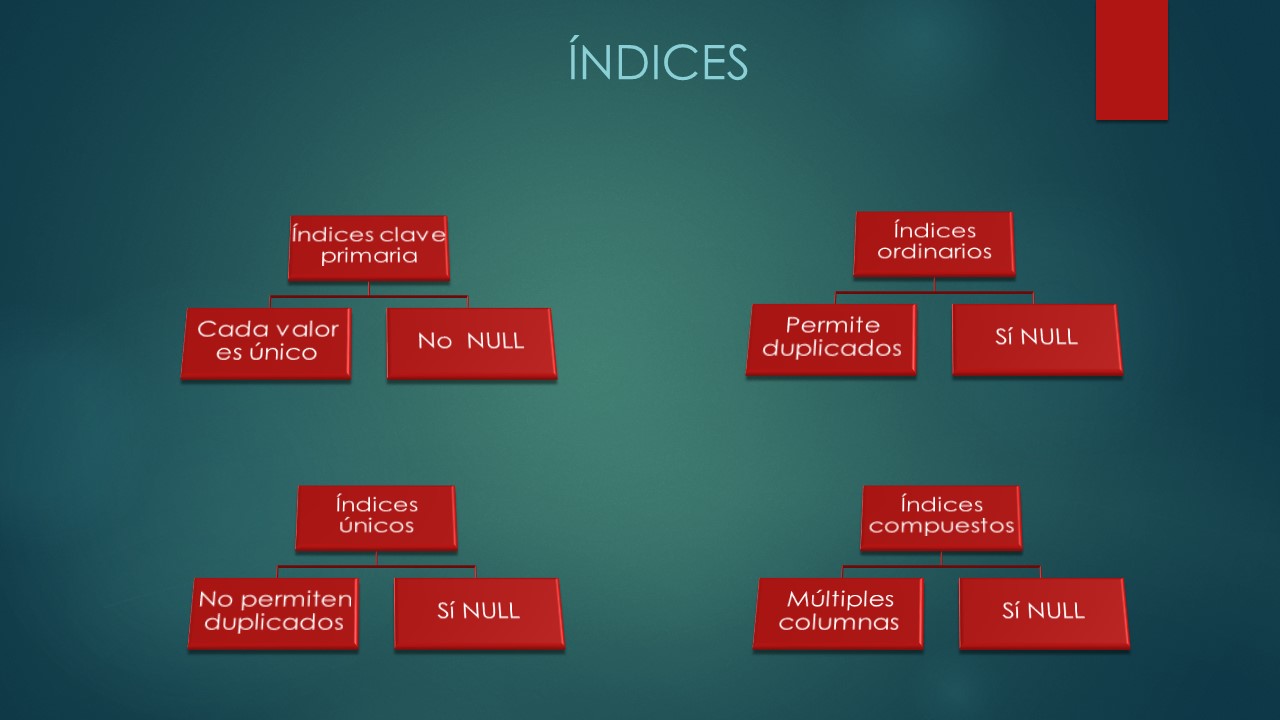
|  |  |
| --- | --- |
| **Tipo de campo** | **Tamaño de Almacenamiento** |
| CHAR(n) | n bytes |
| VARCHAR(n) | n +1 bytes |
| TINYBLOB, TINYTEXT | Longitud+1 bytes |
| BLOB, TEXT | Longitud +2 bytes |
| MEDIUMBLOB, MEDIUMTEXT | Longitud +3 bytes |
| LONGBLOB, LONGTEXT | Longitud +4 bytes |
| ENUM('value1','value2',...) | 1 ó dos bytes dependiendo del número de valores |
| SET('value1','value2',...) | 1, 2, 3, 4 ó 8 bytes, dependiendo del número de valores |

Diferencia de almacenamiento entre los tipos Char y VarChar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Valor** | **CHAR(4)** | **Almacenamiento** | **VARCHAR(4)** | **Almacenamiento** |
| '' | '' | 4 bytes | " | 1 byte |
| 'ab' | 'ab ' | 4 bytes | 'ab' | 3 bytes |
| 'abcd' | 'abcd' | 4 bytes | 'abcd' |  |
| 'abcdefgh' | 'abcd' | 4 bytes | 'abcd' | 5 bytes |

# ÍNDICES..

Los índices nos permiten realizar búsquedas con mayor rapidez en una tabla de una base de datos extensa. Los índices no forman parte del estándar SQL.



Índices de clave primaria:

**ALTER TABLE** EJEMPLO

**ADD PRIMARY KEY**(NOMBRE, APELLIDO)

Índices ordinarios:

**CREATE INDEX** INDICE1

**ON** EJEMPLO(APELLIDO)

Índices únicos:

**CREATE UNIQUE INDEX** INDICE1

**ON** EJEMPLO(APELLIDO)

Índice compuesto:

**CREATE UNIQUE INDEX** INDICE1

**ON** EJEMPLO(NOMBRE, APELLIDO)

Para eliminar un índice:

**DROP INDEX** INDICE1

**ON** EJEMPLO

Si la clave es primaria (en SQL server, Access y Oracle DDBB):

**ALTER TABLE** EJEMPLO

**DROP** **CONSTRAINT** Index\_A37AECA1\_8718\_47A0

En MySQL:

**ALTER TABLE** EJEMPLO

**DROP** **PRIMARY KEY**

Ejemplo de creación de una table con índices:

**CREATE TABLE** miTabla(

Id NUMBER **AUTO\_INCREMENT** **NOT NULL**,

nombre VARCHAR(20) **NOT NULL**,

**CONSTRAINT** pk\_miClave\_primaria **PRIMARY KEY**(id);

**CONSTRAINT** fk\_miClave\_foranea **FOREING KEY**(usuario\_id) **REFERENCES** usuarios(id);

);

# ENGINES

En SQL Podemos usar dos tipos de motor o forma de almacenar los datos:

**- ENGINE=InnoDb**. Por defecto. Me permite mantener la integridad referencial. Si nuestra aplicacion hace muchas consultas de tipo **INSERT** o **UPDATE** se va a notar un aumento del rendimiento.

**-ENGINE=MayISAM**. Me permita tener mayor velocidad a la hora de recuperar datos, es decir, de hacer consultas **SELECT**. No mantiene la integridad referencial.

# TRIGGERS

Los triggers o disparadores, son objetos que se crean en una BBDD, están asociados a una tabla y desencadenarán una acción cuando ocurra algo predefinido, ya sea al insertar (**INSERT**), modificar (**UPDATE**) o borrar registros (**DELETE**).

Se utilizan para tereas de mantenimiento y administración de BBDD.

No se pueden crear en Access.

Es vital determinar cuándo se va a ejecutar el trigger, **antes** o **después** de que se realice la acción. Si queremos almacenar en una tabla los datos del usuario que modifique mi tabla principal, lo tendré que poner **después**.

Si por ejemplo quiero poner un trigger que avise al usuario que al introducir un valor en el campo EDAD, éste no puede ser negativo, lo tendré que poner **antes** de introducir la edad en la tabla.

Si voy a borrar un elemento de mi tabla CLIENTE y esta está tiene integridad referencial con PEDIDOS, primero tendré que comprobar si existe dicha relación, ya que no puede quedar un pedido sin cliente.

Ejemplo, crear un **trigger** asociado a mi tabla PRODUCTOS que introduzca en una nueva tabla después de haber insertado un registro, los datos de dicho registro, CODIGO\_ARTICULO, NOMBRE, PRECIO y FECHA de la modificación.

Primero crearíamos la tabla:

CREATE TABLE REG\_PRODUCTOS(

CODIGO\_ARTICULO VARCHAR(25),

NOMBRE\_ARTICULO VARCHAR(30),

PRECIO DOUBLE(8,2),

FECHA\_INSERCION DATE)

Después el TRIGGER:

**CREATE TRIGGER** PRODUCTOS\_AI

**AFTER INSERT ON** PRODUCTOS **FOR EACH ROW**

**INSERT INTO** reg\_productos(`CODIGO\_ARTICULO`,

`NOMBRE`,

`PRECIO`,

`FECHA\_INSERCION`)

**VALUES**(NEW.`CODIGO `,

NEW.`NOMBRE`,

NEW.`PRECIO`,

NOW())

FOR EACH ROW: Indica que el TRIGGER se va a ejecutar por cada registro insertado.

FOR EACH STATEMENT: Indica que el TRIGGER se va a ejecutar por cada sentencia (que puede contener varios registros).

## TRIGGERS de actualización.

En las actualizaciones de tablas no se puede volver atrás, de modo que si queremos ver la modificación de un artículo debemos crear un TRIGGER con el que hagamos un respaldo de seguridad por ejemplo de los campos precio\_antiguo y precio\_modificado. Este tipo de TRIGGERS han de ejecutarse **antes** de la actualización.

EJ:

**CREATE TABLE** PRODUCTOS\_ACTUALIZADOS(

OLD\_CODIGO\_ARTICULO VARCHAR (4),

OLD\_NOMBRE VARCHAR(20),

OLD\_SECCION VARCHAR(10),

OLD\_PRECIO DOUBLE(8,2),

NEW\_CODIGO\_ARTICULO VARCHAR (4),

NEW\_NOMBRE VARCHAR(20),

NEW\_SECCION VARCHAR(10),

NEW\_PRECIO DOUBLE(8,2),

USUARIO VARCHAR(15),

FECHA\_MOD DATETIME)

TRIGGER:

**CREATE TRIGGER** DATOS\_CAMBIO\_BI

**BEFORE UPDATE**

**ON** productos

**FOR EACH ROW**

**INSERT INTO** productos\_actualizados(

OLD\_CODIGO\_ARTICULO,

OLD\_SECCION,

OLD\_NOMBRE,

OLD\_PRECIO,

NEW\_CODIGO\_ARTICULO,

NEW\_SECCION,

NEW\_NOMBRE,

NEW\_PRECIO,

USUARIO,

FECHA\_MOD)

**VALUES**(

OLD.CODIGO\_ARTICULO,

OLD.NOMBRE,

OLD.SECCION,

OLD.PRECIO,

NEW.CODIGO\_ARTICULO,

NEW.NOMBRE,

NEW.SECCION,

NEW.PRECIO,

CURRENT\_USER(),

NOW())

POR ÚLTIMO, REALIZAMOS LA CONSULTA:

**UPDATE** productos

**SET** PRECIO=PRECIO+10

**WHERE** CODIGO\_ARTICULO='AR99'



# APÉNDICE FUNCIONES SQL

El **lenguaje SQL** tiene **funciones incorporadas para hacer cálculos sobre los datos**. Las funciones se pueden dividir en dos grupos (existen muchas mas, que dependen del **sistema de bases de datos** que se utilice):

**Funciones agregadas SQL**, devuelven un sólo valor, calculado con los valores de una columna.

1. AVG() - La **media de los valores**
2. COUNT() - El **número de filas**
3. MAX() - El **valor más grande**
4. MIN() - El **valor más pequeño**
5. SUM() - La **suma de los valores**
6. GROUP BY - Es una **sentencia** que va muy ligada a las funciones agregadas

**Funciones escalares SQL**, devuelven un sólo valor basándose en el valor de entrada.

1. UCASE() - Convierte un campo a **mayúsculas**
2. LCASE() - Convierte un campo a **minúsculas**
3. MID() - **Extrae** caracteres de un campo de texto
4. LEN() - Devuelve la **longitud** de un campo de texto
5. NOW() - Devuelve la **hora y fecha** actuales del sistema
6. FORMAT() - Da formato a un **formato** para mostrarlo

Para la explicación de las funciones, una de las tablas con las que vamos a trabajar es **Productos**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ProductoID** | **NombreProducto** | **Descripcion** | **Precio** | **Stock** |
| 1 | Camiseta | Camiseta negra simple de talla única | 10 | 16 |
| 2 | Pantalón | Pantalón largo azul tipo chino | 20 | 24 |
| 3 | Gorra | Gorra azul con el logo de los Yankees | 15 | 32 |
| 4 | Zapatillas | Zapatillas de running de color blanco y verde | 35 | 13 |

## Funciones agregadas SQL

### 1. AVG()

La función ***AVG()*** devuelve la **media de valores de una columna numérica**.

SELECT AVG (nombreColumna) FROM nombreTabla;

La siguiente sentencia SQL devuelve la media del stock que hay en el almacén:

SELECT AVG (Stock) FROM Productos;

La siguiente sentencia muestra el NombreProducto y el Precio de los registros que tienen un Precio por encima de la media:

SELECT NombreProducto, Precio FROM Productos

WHERE Precio > (SELECT AVG (Precio) FROM Productos);

### 2. COUNT()

La función ***COUNT()*** devuelve el **número de filas** que cumplen con un determinado criterio:

**Número de valores en una columna** (los valores NULL no se cuentan):

SELECT COUNT (nombreColumna) FROM nombreTabla;

**Número de registros en una tabla**

SELECT COUNT(\*) FROM nombreTabla;

**Número de valores distintos de una columna**

SELECT COUNT (DISTINCT nombreColumna) FROM nombreTabla;

Vamos a emplear ahora la tabla **Pedidos**:

|  |  |  |
| --- | --- | --- |
| **PedidoID** | **ClienteID** | **Factura** |
| 234 | 4 | 160 |
| 235 | 5 | 48 |
| 236 | 12 | 64 |
| 237 | 4 | 92 |

Si queremos conocer el **número de pedidos del cliente número 4**:

SELECT COUNT (ClienteID) AS PedidosCliente4 FROM Pedidos

WHERE ClienteID=4;

Si queremos simplemente contar el **número de pedidos totales**:

SELECT COUNT(\*) AS PedidosTotales FROM Pedidos;

Si queremos contar el **número de clientes únicos** desde la tabla Pedidos:

SELECT COUNT (DISTINCT ClienteID) AS NumeroClientes FROM Pedidos;

### 3. MAX()

La función ***MAX()*** devuelve el **mayor valor de la columna seleccionada**:

SELECT MAX (nombreColumna) FROM nombreTabla;

De la tabla productos vamos a coger **el producto más caro**:

SELECT MAX (Precio) AS ProductoMasCaro FROM Productos;

### 4. MIN()

La función ***MIN()*** devuelve el **menor valor de la columna seleccionada**:

SELECT MIN (nombreColumna) FROM nombreTabla;

De la tabla productos vamos a coger **el producto más barato**:

SELECT MIN (Precio) AS ProductoMasBarato FROM Productos;

### 5. SUM()

La función ***SUM()*** devuelve la **suma de una columna numérica**:

SELECT SUM (nombreColumna) FROM nombreTabla;

De la tabla productos vamos a coger el **número total de productos en stock**:

SELECT SUM (Stock) AS ProductosTotales FROM Productos;

### 6. GROUP BY

La sentencia **GROUP BY** se utiliza junto con las funciones agregadas para **agrupar en un result-set una o más columnas**.

SELECT nombreColumna, funcion\_agregada(nombreColumna)

FROM nombreTabla

WHERE nombreColumna operador valor

GROUP BY nombreColumna;

Si tenemos las tablas **Productos**, **Pedidos** y **Clientes**:

**Productos**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ProductoID** | **NombreProducto** | **Descripcion** | **Precio** | **Stock** |
| 1 | Camiseta | Camiseta negra simple de talla única | 10 | 16 |
| 2 | Pantalón | Pantalón largo azul tipo chino | 20 | 24 |
| 3 | Gorra | Gorra azul con el logo de los Yankees | 15 | 32 |
| 4 | Zapatillas | Zapatillas de running de color blanco y verde | 35 | 13 |

**Pedidos**:

|  |  |  |
| --- | --- | --- |
| **PedidoID** | **ClienteID** | **Factura** |
| 234 | 4 | 160 |
| 235 | 5 | 48 |
| 236 | 12 | 64 |
| 237 | 4 | 92 |

**Clientes**:

|  |  |  |
| --- | --- | --- |
| **ClienteID** | **NombreCliente** | **Contacto** |
| 1 | Lorena Higgins | 456443552 |
| 2 | Raúl González | 445332221 |
| 3 | Carmen Smith | 488982635 |
| 4 | Fernando Stewart | 412436773 |

Vamos a obtener el **número de pedidos** realizados **por cada cliente**:

SELECT Clientes.NombreCliente, Count(Pedidos.PedidoID) AS NumeroPedidos FROM Pedidos

LEFT JOIN Clientes

ON Pedidos.ClienteID=Clientes.ClienteID

GROUP BY NombreCliente;

Podemos **utilizar GROUP BY en más de una columna**:

SELECT Clientes.NombreCliente, Productos.NombreProducto, COUNT (Pedidos.PedidoID) AS NumeroPedidos

FROM ((Pedidos

INNER JOIN Clientes ON Pedidos.ClienteID=Clientes.ClienteID)

INNER JOIN Productos ON Pedidos.ProductoID=Productos.ProductoID)

GROUP BY NombreCliente, NombreProducto;

## Funciones escalares SQL

### 7. UCASE()

La función ***UCASE()*** convierte el **valor de un campo a mayúsculas**.

SELECT UCASE (nombreColumna) FROM nombreTabla;

Si queremos obtener todos los **nombres de los clientes en mayúsculas**:

SELECT UCASE (NombreCliente) AS Cliente FROM Clientes;

### 8. LCASE()

La función ***LCASE()*** convierte el **valor de un campo en minúsculas**:

SELECT LCASE (nombreColumna) FROM nombreTabla;

Si queremos obtener todos los **nombres de los clientes en minúsculas**:

SELECT LCASE (NombreCliente) FROM Clientes;

### 9. MID()

La función ***MID()*** se usa para **extraer caracteres desde un campo de texto**.

SELECT MID (nombreColumna, inicio [, longitud]) AS unAlias FROM nombreTabla;

Los campos obligatorios son ***nombreColumna*** e ***inicio*** (especifica la posición donde comenzar la extracción). El campo opcional es ***longitud***, que especifica el número de caracteres a devolver. Si se omite, devuelve todos los caracteres hasta el final del texto.

Vamos a extraer por ejemplo los **primeros cuatro caracteres** de **NombreProducto** de la tabla **Productos**:

SELECT MID (nombreProducto, 1, 4) AS ProductoCorto

From Productos;

### 10. LEN()

La función ***LEN()*** devuelve la longitud de un campo de texto.

SELECT LEN (nombreColumna) FROM nombreTabla;

Si queremos obtener **nombreProducto** y la **longitud de las descripciones** de los productos de la tabla **Productos**:

SELECT NombreProducto, LEN(Descripcion) as LongitudDescripcion

FROM Productos;

### 11. NOW()

La función ***NOW()*** devuelve la hora y fecha actuales.

SELECT NOW() FROM nombreTabla;

Si queremos devolver el **nombreProducto** y **Precio** de hoy de la tabla **Productos**:

SELECT NombreProducto, Precio, Now() AS PrecioProductosHoy FROM Productos;

### 12. FORMAT()

La función ***FORMAT()*** se usa para formatear cómo se mostrará un campo.

SELECT FORMAT(nombreColumna, formato) FROM nombreTabla;

Vamos a mostrar un **formato de fecha del ejemplo anterior**:

SELECT NombreProducto, Precio, FORMAT(Now(), 'YYYY-MM-DD') AS PrecioProductosHoy FROM Productos;

## FUNCIONES ARITMÉTICAS.

**Función SQL ABS ()**

El SQL ABS () devuelve el valor absoluto de un número pasado como argumento.

Sintaxis: ABS (Expresión)

**SELECT** **ABS** (-17.36)

**FROM** dual;

Salida: 17.36

**Función SQL CEIL ()**

El SQL CEIL () redondeará hacia arriba cualquier valor decimal positivo o negativo dentro de la función hacia arriba

Sintaxis: CEIL (Expresión)

**SELECT** **CEIL** (17.36)

**FROM** dual;

Salida: 18

**SELECT** **CEIL** (-17.36);

Salida: -17

**Función SQL FLOOR ()**

El FLOOR de SQL () redondea hacia arriba cualquier valor decimal positivo o negativo hasta el siguiente valor entero menor. SQL DISTINCT, junto con la función SQL FLOOR (), se usa para recuperar solo un valor único después de redondearlo al siguiente valor entero menor, dependiendo de la columna especificada.

Sintaxis: FLOOR (Expresión)

**SELECT FLOOR** (17.36)

**FROM** dual;

Salida: 17

**SELECT FLOOR** (-17.36)

**FROM** dual;

Salida: -18

**Función SQL EXP ()**

El SQL EXP () devuelve e elevado a la potencia n-th (n es la expresión numérica), donde **e** es la base del algoritmo natural y el valor de e es aproximadamente 2.71828183.

Sintaxis: EXP (Expresión)

**SELECT EXP** (2) AS e\_to\_2s\_power

**FROM** dual;

Salida: 7.3890560989306502272304274605750078132

**Función SQL LN ()**

La función SQL LN () devuelve el logaritmo natural de n, donde n es mayor que 0 y su base es un número igual a aproximadamente 2.71828183.

Sintaxis: LN (Expresión)

**SELECT LN** (65) "natural\_log de 65"

**FROM** dual;

Salida: 4.17438726989563711065424677479150624431

**Función SQL MOD ()**

La función SQL MOD () devuelve el resto de una división. El comando DISTINCT de SQL junto con la función MOD () de SQL se usa para recuperar solo registros únicos según la columna o expresión especificada.

Sintaxis: MOD (dividendo, divisor)

**SELECT MOD** (25, 7)

**FROM** dual;

Salida: 4

**SELECT MOD** (-25, 7)

**FROM** dual;

Salida: -4

**Función SQL POWER ()**

La función SQL POWER () devuelve el valor de un número elevado a otro, donde los dos números se pasan como argumentos. El comando DISTINCT de SQL junto con la función POWER () de SQL se puede usar para recuperar solo datos únicos en función de una expresión específica.

Sintaxis: POWER (base, exponente)

**SELECT POWER** (2, 3)

**FROM** dual;

Salida: 8

**Función SQL SQRT ()**

El SQL SQRT () devuelve la raíz cuadrada del valor dado en el argumento.

Sintaxis: SQRT (expresión)

**SELECT SQRT** (36)

**FROM** dual;

Salida: 6

## FUNCIONES PARA TEXTO.

En MySQL tenemos muchas funciones para manipular cadenas: calcular su longitud, extraer un fragmento situado a la derecha, a la izquierda o en cualquier posición, eliminar espacios finales o iniciales, convertir a hexadecimal y a binario, etc. Vamos a comentar las más habituales. Los ejemplos estarán aplicados directamente sobre cadenas, pero (por supuesto) también se pueden aplicar a campos de una tabla:

**10.1. Funciones de conversión a mayúsculas/minúsculas**

LOWER o LCASE convierte una cadena a minúsculas: SELECT LOWER('Hola'); ⇒ hola

UPPER o UCASE convierte una cadena a mayúsculas: SELECT UPPER('Hola'); ⇒ HOLA

**10.2. Funciones de extracción de parte de la cadena**

LEFT(cadena, longitud) extrae varios caracteres del comienzo (la parte izquierda) de la cadena: SELECT LEFT('Hola',2); ⇒ Ho

RIGHT(cadena, longitud) extrae varios caracteres del final (la parte derecha) de la cadena: SELECT RIGHT('Hola',2); ⇒ la

MID(cadena, posición, longitud), SUBSTR(cadena, posición, longitud) o SUBSTRING(cadena, posición, longitud) extrae varios caracteres de cualquier posición de una cadena, tantos como se indique en "longitud": SELECT SUBSTRING('Hola',2,3); ⇒ ola (Nota: a partir MySQL 5 se permite un valor negativo en la posición, y entonces se comienza a contar desde la derecha -el final de la cadena-)

CONCAT une (concatena) varias cadenas para formar una nueva: SELECT CONCAT('Ho', 'la'); ⇒ Hola

CONCAT\_WS une (concatena) varias cadenas para formar una nueva, usando un separador que se indique (With Separator): SELECT CONCAT\_WS('-','Ho','la','Que','tal'); ⇒ Ho-la-Que-tal

LTRIM devuelve la cadena sin los espacios en blanco que pudiera contener al principio (en su parte izquierda): SELECT LTRIM(' Hola'); ⇒ Hola

RTRIM devuelve la cadena sin los espacios en blanco que pudiera contener al final (en su parte derecha): SELECT RTRIM('Hola '); ⇒ Hola

TRIM devuelve la cadena sin los espacios en blanco que pudiera contener al principio ni al final: SELECT TRIM(' Hola '); ⇒ Hola (Nota: realmente, TRIM puede eliminar cualquier prefijo, no sólo espacios; mira el manual de MySQL para más detalles)

**10.3. Funciones de conversión de base numérica**

BIN convierte un número decimal a binario: SELECT BIN(10); ⇒ 1010

HEX convierte un número decimal a hexadecimal: SELECT HEX(10); ⇒ 'A' (Nota: HEX también tiene un uso alternativo menos habitual: puede recibir una cadena, y entonces mostrará el código ASCII en hexadecimal de sus caracteres: SELECT HEX('Hola'); ⇒ '486F6C61')

OCT convierte un número decimal a octal: SELECT OCT(10); ⇒ 12

CONV(número,baseInicial,baseFinal) convierte de cualquier base a cualquier base: SELECT CONV('F3',16,2); ⇒ 11110011

UNHEX convierte una serie de números hexadecimales a una cadena ASCII, al contrario de lo que hace HEX: SELECT UNHEX('486F6C61'); ⇒ 'Hola')

**10.4. Otras funciones de modificación de la cadena**

INSERT(cadena,posición,longitud,nuevaCadena) inserta en la cadena otra cadena: SELECT INSERT('Hola', 2, 2, 'ADIOS'); ⇒ HADIOSa

REPLACE(cadena,de,a) devuelve la cadena pero cambiando ciertas secuencias de caracteres por otras: SELECT REPLACE('Hola', 'l', 'LLL'); ⇒ HoLLLa

REPEAT(cadena,numero) devuelve la cadena repetida varias veces: SELECT REPEAT(' Hola',3); ⇒ HolaHolaHola

REVERSE(cadena) devuelve la cadena "del revés": SELECT REVERSE('Hola'); ⇒ aloH

SPACE(longitud) devuelve una cadena formada por varios espacios en blanco: SELECT SPACE(3); ⇒ "   "

**10.5. Funciones de información sobre la cadena**

CHAR\_LENGTH o CHARACTER\_LENGTH devuelve la longitud de la cadena en caracteres

LENGTH devuelve la longitud de la cadena en bytes

BIT\_LENGTH devuelve la longitud de la cadena en bits

INSTR(cadena,subcadena) o LOCATE(subcadena,cadena,posInicial) devuelve la posición de una subcadena dentro de la cadena: SELECT INSTR('Hola','ol'); ⇒ 2

## FUNCIONES PARA FECHAS.

|  |  |
| --- | --- |
| [ADDDATE](http://mysql.conclase.net/curso/?sqlfun=ADDDATE) | Suma un intervalo de tiempo a una fecha |
| [ADDTIME](http://mysql.conclase.net/curso/?sqlfun=ADDTIME) | Suma tiempos |
| [CONVERT\_TZ](http://mysql.conclase.net/curso/?sqlfun=CONVERT_TZ) | Convierte tiempos entre distintas zonas horarias |
| [CURDATE o CURRENTDATE](http://mysql.conclase.net/curso/?sqlfun=CURDATE) | Obtener la fecha actual |
| [CURTIME o CURRENT\_TIME](http://mysql.conclase.net/curso/?sqlfun=CURTIME) | Obtener la hora actual |
| [DATE](http://mysql.conclase.net/curso/?sqlfun=DATE) | Extraer la parte correspondiente a la fecha |
| [DATEDIFF](http://mysql.conclase.net/curso/?sqlfun=DATEDIFF) | Calcula la diferencia en días entre dos fechas |
| [DATE\_ADD](http://mysql.conclase.net/curso/?sqlfun=DATE_ADD) | Aritmética de fechas, suma un intervalo de tiempo |
| [DATE\_SUB](http://mysql.conclase.net/curso/?sqlfun=DATE_ADD) | Aritmética de fechas, resta un intervalo de tiempo |
| [DATE\_FORMAT](http://mysql.conclase.net/curso/?sqlfun=DATE_FORMAT) | Formatea el valor de una fecha |
| [DAY o DAYOFMONTH](http://mysql.conclase.net/curso/?sqlfun=DAYOFMONTH) | Obtiene el día del mes a partir de una fecha |
| [DAYNAME](http://mysql.conclase.net/curso/?sqlfun=DAYNAME) | Devuelve el nombre del día de la semana |
| [DAYOFWEEK](http://mysql.conclase.net/curso/?sqlfun=DAYOFWEEK) | Devuelve el índice del día de la semana |
| [DAYOFYEAR](http://mysql.conclase.net/curso/?sqlfun=DAYOFYEAR) | Devuelve el día del año para una fecha |
| [EXTRACT](http://mysql.conclase.net/curso/?sqlfun=EXTRACT) | Extrae parte de una fecha |
| [FROM\_DAYS](http://mysql.conclase.net/curso/?sqlfun=FROM_DAYS) | Obtener una fecha a partir de un número de días |
| [FROM\_UNIXTIME](http://mysql.conclase.net/curso/?sqlfun=FROM_UNIXTIME) | Representación de fechas UNIX en formato de cadena |
| [GET\_FORMAT](http://mysql.conclase.net/curso/?sqlfun=GET_FORMAT) | Devuelve una cadena de formato |
| [HOUR](http://mysql.conclase.net/curso/?sqlfun=HOUR) | Extrae la hora de un valor time |
| [LAST\_DAY](http://mysql.conclase.net/curso/?sqlfun=LAST_DAY) | Devuelve la fecha para el último día del mes de una fecha |
| [MAKEDATE](http://mysql.conclase.net/curso/?sqlfun=MAKEDATE) | Calcula una fecha a partir de un año y un día del año |
| [MAKETIME](http://mysql.conclase.net/curso/?sqlfun=MAKETIME) | Calcula un valor de tiempo a partir de una hora, minuto y segundo |
| [MICROSECOND](http://mysql.conclase.net/curso/?sqlfun=MICROSECOND) | Extrae los microsegundos de una expresión de fecha/hora o de hora |
| [MINUTE](http://mysql.conclase.net/curso/?sqlfun=MINUTE) | Extrae el valor de minutos de una expresión time |
| [MONTH](http://mysql.conclase.net/curso/?sqlfun=MONTH) | Devuelve el mes de una fecha |
| [MONTHNAME](http://mysql.conclase.net/curso/?sqlfun=MONTHNAME) | Devuelve el nombre de un mes para una fecha |
| [NOW o CURRENT\_TIMESTAMP o LOCALTIME o LOCALTIMESTAMP o SYSDATE](http://mysql.conclase.net/curso/?sqlfun=NOW) | Devuelve la fecha y hora actual |
| [PERIOD\_ADD](http://mysql.conclase.net/curso/?sqlfun=PERIOD_ADD) | Añade meses a un periodo (año/mes) |
| [PERIOD\_DIFF](http://mysql.conclase.net/curso/?sqlfun=PERIOD_DIFF) | Calcula la diferencia de meses entre dos periodos (año/mes) |
| [QUARTER](http://mysql.conclase.net/curso/?sqlfun=QUARTER) | Devuelve el cuarto del año para una fecha |
| [SECOND](http://mysql.conclase.net/curso/?sqlfun=SECOND) | Extrae el valor de segundos de una expresión time |
| [SEC\_TO\_TIME](http://mysql.conclase.net/curso/?sqlfun=SEC_TO_TIME) | Convierte una cantidad de segundos a horas, minutos y segundos |
| [STR\_TO\_DATE](http://mysql.conclase.net/curso/?sqlfun=STR_TO_DATE) | Obtiene un valor DATETIME a partir de una cadena con una fecha y una cadena de formato |
| [SUBDATE](http://mysql.conclase.net/curso/?sqlfun=SUBDATE) | Resta un intervalo de tiempo de una fecha |
| [SUBTIME](http://mysql.conclase.net/curso/?sqlfun=SUBTIME) | Resta dos expresiones time |
| [TIME](http://mysql.conclase.net/curso/?sqlfun=TIME) | Extrae la parte de la hora de una expresión fecha/hora |
| [TIMEDIFF](http://mysql.conclase.net/curso/?sqlfun=TIMEDIFF) | Devuelve en tiempo entre dos expresiones de tiempo |
| [TIMESTAMP](http://mysql.conclase.net/curso/?sqlfun=TIMESTAMP) | Convierte una expresión de fecha en fecha/hora o suma un tiempo a una fecha |
| [TIMESTAMPADD](http://mysql.conclase.net/curso/?sqlfun=TIMESTAMPADD) | Suma un intervalo de tiempo a una expresión de fecha/hora |
| [TIMESTAMPDIFF](http://mysql.conclase.net/curso/?sqlfun=TIMESTAMPDIFF) | Devuelve la diferencia entre dos expresiones de fecha/hora |
| [TIME\_FORMAT](http://mysql.conclase.net/curso/?sqlfun=TIME_FORMAT) | Formatea un tiempo |
| [TIME\_TO\_SEC](http://mysql.conclase.net/curso/?sqlfun=TIME_TO_SEC) | Convierte un tiempo a segundos |
| [TO\_DAYS](http://mysql.conclase.net/curso/?sqlfun=TO_DAYS) | Calcula el número de días desde el año cero |
| [UNIX\_TIMESTAMP](http://mysql.conclase.net/curso/?sqlfun=UNIX_TIMESTAMP) | Devuelve un timestamp o una fecha en formato UNIX, segundos desde 1070 |
| [UTC\_DATE](http://mysql.conclase.net/curso/?sqlfun=UTC_DATE) | Devuelve la fecha UTC actual |
| [UTC\_TIME](http://mysql.conclase.net/curso/?sqlfun=UTC_TIME) | Devuelve la hora UTC actual |
| [UTC\_TIMESTAMP](http://mysql.conclase.net/curso/?sqlfun=UTC_TIMESTAMP) | Devuelve la fecha y hora UTC actual |
| [WEEK](http://mysql.conclase.net/curso/?sqlfun=WEEK) | Calcula el número de semana para una fecha |
| [WEEKDAY](http://mysql.conclase.net/curso/?sqlfun=WEEKDAY) | Devuelve el número de día de la semana para una fecha |
| [WEEKOFYEAR](http://mysql.conclase.net/curso/?sqlfun=WEEKOFYEAR) | Devuelve el número de la semana del año para una fecha |
| [YEAR](http://mysql.conclase.net/curso/?sqlfun=YEAR) | Extrae el año de una fecha |
| [YEARWEEK](http://mysql.conclase.net/curso/?sqlfun=YEARWEEK) | Devuelve el año y semana de una fecha |

## FUNCIONES GENERALES.

Algunas funciones generales de SQL son:

Para comprobar si un campo es nulo, me devuelve 0 (false) si no es nulo o 1 (true) si es nulo:

**SELECT** **ISNULL**(NOMBRE)

**FROM** USUARIOS

Para comprobar si dos campos son diferentes, devuelve **1** (**true)** si son diferentes o **0** (**false)** si son iguales:

**SELECT** **STRCMP**(NOMBRE1, NOMBRE2)

**FROM** USUARIOS

Para ver la versión de MySQL que tengo:

**SELECT** **VERSION()**

**FROM** USUARIOS.

Para ver que usuario tengo a nivel de **SGBD**, es decir, el usuario que tengo dado de alta y con el cual me he identificado para trabajar:

**SELECT** **USER()**

**FROM** USUARIOS

Para saber el nombre de la BBDD que estoy usando:

**SELECT** **DATABASE()**

**FROM** USUARIOS

Para rellenar un campo en caso de que esté vacío:

**SELECT** **IFFNULL(**nombre, ‘Este campo está vacío’**)**

**FROM** USUARIOS