

# SQL: INSERTAR DATOS

## INSERTAR DATOS EN UNA TABLA

```
INSERT INTO <nombre_tabla> (<campo1>,...<campoN>)  
VALUES ( 'valor1', ..., 'valorN');
```

-Es muy importante que el numero de campos - valores **coincida**.

-Los valores han de ir entre comillas simples ` ` si es texto.

Comillas simples: si es texto, varchar, date  
Sin comillas: Numeros

No se han puesto los campos, se entran los datos  
en el mismo orden de la estructura de datos

Entrada parcial, los campos que no ponemos  
datos han de permitir valores nulos

# SQL: INSERTAR DATOS

La sentencia utilizada para insertar nuevos datos en la BBDD es INSERT. Su formato puede variar entre las siguientes formas, en función de si especificamos un valor para todas las columnas o no:

```
INSERT INTO alumnos VALUES (NULL, 'José Martínez', '1980-05-13', 'Calle Central 17', '678564321');
```

```
INSERT INTO alumnos (nombre, fecha_nacimiento) VALUES ('José Martínez', '1980-05-13');
```

```
INSERT INTO alumnos SET nombre='José Martínez',  
fecha_nacimiento='1980-05-13';
```

En la primera sentencia, el valor **NULL** se utiliza en la primera columna, normalmente el identificador, que normalmente se insertará de manera automática gracias a que suelen ser del tipo **AUTO\_INCREMENT**.

# SQL: ACTUALIZAR DATOS

## ACTUALIZAR DATOS EN UNA TABLA

**UPDATE** <nombre\_tabla>

**SET** <campo1> = <valor1>, ..., <campoN> = <valorN>

**WHERE** <condicion>;

IMPORTANTE: Hemos de poner una codicion. Si no, nos cambiara toda la base de datos

Hola, als apunts que hem estat seguint a classe hi havia un error.  
Concretament a la secció d'updates amb subqueries -pag 3.2, exemple 4.3-.

Actualment mysql no accepta aquesta sintaxi. Us fico un exemple de com fer-ho:

Per exemple, per canviar el nom i el carrec del empleat amb 109 de manera que siguin idèntics als de l'empleat 101.

```
UPDATE empleados as e, (select * from empleados where id_empleado =101) as ee  
SET e.cargo=ee.cargo, e.nombre=ee.nombre where e.id_empleado=109;
```

Salutacions i perdoneu les molèsties.  
Marc.

# SQL: ACTUALIZAR DATOS

## ACTUALIZAR DATOS EN UNA TABLA

**WHERE** <condiciones\_boleanas>;

-Se utilizan los operadores =, <, <=, >, >=, <>, NOT, AND, OR

-Por ejemplo:

```
UPDATE EMPLEADOS SET sueldo = 1000, oficina=3  
WHERE nombre='Pepito';
```

# SQL: BORRRAR DATOS

## BORRRAR DATOS EN UNA TABLA

### DELETE

**FROM** <nombre\_tabla>

**WHERE** <condicion>;

IMPORTANTE: Si no ponemos  
WHERE, vaciamos la tabla

DELETE FROM empleados WHERE numemp='C2';  
//borraria el registro con número de empleado C2

# SQL: ACTUALIZAR DATOS

La sentencia utilizada para modificar datos ya existentes en la BBDD es UPDATE. Su formato es el siguiente:

```
UPDATE alumnos SET sexo=0 WHERE nombre='pedro';
```

Para eliminar los datos, usaremos la sentencia DELETE:

```
DELETE FROM alumnos WHERE nombre='pedro';
```

Hay que ir con **MUCHO CUIDADO** con estas dos sentencias, ya que los cambios realizados son irreversibles, y por tanto no se pueden utilizar a la ligera. Hay que indicar que, **en caso de no añadir la clausula WHERE y ejecutarlas, se modificarían o eliminarían TODOS los registros de la tabla**, con lo que ello pueda conllevar.

# SQL: CONSULTAR DATOS

## CONSULTAR DATOS EN UNA TABLA

Elimina duplicados en al visualizacion

Apodo

```
SELECT [DISTINCT] <nombre_columnas> [as nuevo_nombre]  
FROM <nombre_tabla>;
```

- Se pueden seleccionar todos los campos con el símbolo \*
- Podremos hacer operaciones directamente en las consultas select.

```
SELECT numero_empleado, (ventas-gastos) AS beneficios FROM  
EMPLEADOS;
```

```
SELECT * FROM EMPLEADOS;
```

Este campo no existe, es un apodo que se visualizara en el lugar del campo

# SQL: CONSULTAR DATOS

## SELECCIONAR DATOS DE UNA TABLA FILTRADOS

```
SELECT [DISTINCT] <nombre_columnas> [as nuevo_nombre]  
FROM <nombre_tabla> WHERE <condiciones_booleanas>;
```

-Se utilizan los operadores =, <, <=, >, >=, <>, NOT, AND, OR

```
SELECT numero_empleado  
FROM EMPLEADOS  
WHERE nombre='Pepito';
```



# SQL: CONSULTAR DATOS

## SELECCIONAR DATOS DE UNA TABLA FILTRADOS

- Para **valores alfanuméricos** podemos utilizar

<campo> [ NOT] BETWEEN <limite1> AND <limite2> Entre - Rango

<campo> [NOT ] IN (<valor1>,...<valorN>) Dentro de la relacion

- Para **valores de tipo cadena** podemos utilizar

<campo> [ NOT] LIKE <expresión> //es como usar el = Valor en concreto

Si usamos LIKE en las cadenas se nos permite usar varios comodines interesantes:

**% encaja con cualquier texto.** (P.E: LIKE "ho%")

**\_ encaja con cualquier carácter** (P.E: LIKE "ho\_")

# SQL: CONSULTAR DATOS

## SELECCIONAR DATOS DE UNA TABLA FILTRADOS

Para consultar si un valor es o no nulo:

<campo> IS [NOT] NULL

- Nunca usar <campo> <> NULL

# SQL: CONSULTAR DATOS

## SELECCIONAR DATOS DE UNA TABLA ORDENADOS

Elimina duplicados en al visualizacion

Apodo

```
SELECT [DISTINCT] <nombre_columnas> [as nuevo_nombre]  
FROM <nombre_tabla> [WHERE <condiciones_booleanas>]  
ORDER BY <campo1>, ... <campo N> [ASC|DESC];
```

```
SELECT *  
FROM EMPLEADOS  
ORDER BY sueldo DESC;
```

- Por defecto el valor será ASC

# SQL: CONSULTAR DATOS

## SELECCIONAR DATOS DE UNA TABLA LIMITADOS

Limita la cantidad de registros a mostrar  
Empezando por...  
Cantidad a enseñar "empezar"

```
SELECT [DISTINCT] <nombre_columnas> [as nuevo_nombre]  
FROM <nombre_tabla> [WHERE <condiciones_booleanas>]  
LIMIT empezar, cantidad;
```

- empezar** es el valor por el cual empezara a recoger los registros.
- cantidad** es la cantidad de registros que recoge

Ejemplo:

```
SELECT * FROM `your_table` LIMIT 5, 5
```

-Si hay 1,2,...10 registros, esta consulta recoge el registro 6, 7, 8, 9 y 10.

# SQL: CONSULTAR DATOS

## SELECCIONAR DATOS DE UNA TABLA CON FUNCIONES

Funciones:

- COUNT(<campo1>) //cuenta cuantas filas hay
- SUM(<campo1>) //suma los valores de una columna
- MIN(<campo1>) //devuelve el mínimo una columna
- MAX(<campo1>) // devuelve el máximo de una columna
- AVG(<campo1>) // devuelve la media de una columna

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

//Devuelve cuantos hay en total


```
SELECT AVG(sueldo)  
FROM EMPLEADOS;
```

//Devuelve la media de sueldo

# SQL: CONSULTAR DATOS

## SELECCIONAR DATOS DE UNA TABLA AGRUPADOS

```
SELECT [DISTINCT] <nombre_columnas> [as nuevo_nombre]  
FROM <nombre_tabla> [WHERE <condiciones_booleanas>]  
GROUP BY <campo1>, ... <campo N>;
```

 Agrupa los campos con el mismo contenido (Repetidos)

Nota: las columnas agrupadas deben de aparecer en la SELECT

```
SELECT oficina, AVG(sueldo)  
FROM EMPLEADOS  
GROUP BY oficina;
```

//nos devuelve la media de sueldo para cada oficina

# SQL: CONSULTAR DATOS

## SELECCIONAR DATOS DE UNA TABLA AGRUPADOS

```
SELECT [DISTINCT] <nombre_columnas> [as nuevo_nombre]  
FROM <nombre_tabla> [WHERE <condiciones_booleanas>]  
GROUP BY <campo1>, ... <campo N>  
HAVING <condición_para_los_grupos>;
```

Esquivala a un WHERE de GRUPOS

Nota: HAVING solo tiene sentido si existe un GROUP BY previo

```
SELECT oficina, AVG(sueldo)  
FROM EMPLEADOS  
GROUP BY oficina  
HAVING AVG(sueldo) < 1000
```

//nos devuelve la media de sueldo para cada oficina si hay 2 o más

# SQL: CONSULTAR DATOS

## RESUMEN DE SELECT

**SELECT** [DISTINCT] <nombre\_columnas> [as nuevo\_nombre]  
**FROM** <nombre\_tabla>

[ **WHERE** <condiciones\_booleanas> ]

← **WHERE** es solo para **SELECT**

[**GROUP BY** <campo1>, ... <campo N>  
**HAVING** <condición\_para\_los\_grupos> ]

← **HAVING** es solo para **GROUP BY**

[**ORDER BY** <campo1>, ... <campoN> ]

[**LIMIT** empieza, cantidad];



# SQL: CONSULTAS DE VARIAS TABLAS

- Hasta ahora se ha visto como consultar datos de una sola tabla.
- Si se han añadido varias tablas y claves foráneas entre ellas lo más normal es querer filtrar y consultar datos que estén presentes en **ambas a la vez**.
- Por ejemplo: Consultar las compras que ha hecho todo un usuario.
- Para ello existen varias formas de hacerlo:
  - UNIONES NATURALES**
  - JOINS**
  - SUBCONSULTAS**

# SQL: UNIONES NATURALES

Esta es una de las formas más utilizadas y la forma más fácil de cruzar datos entre dos tablas:

-Se deben identificar las dos tablas separadas por comas en FROM y usar esta notación para evitar problemas entre campos que se llamen igual en las dos tablas.

**union natural:** solo se muestran los campos si estan en las dos tablas

FROM COMANDA c, LINIA\_COMANDA lc

- Sabiendo esto, también se debe identificar los campos que queremos de cada una de las tablas:

SELECT c.numcomanda, lc.lin\_com, lc.codfab, lc.codprod

# SQL: UNIONES NATURALES

-El **WHERE** se encarga de relacionar las dos tablas mediante sus **claves foráneas**:

WHERE c.numcomanda=lc.numcomanda;

Ejemplo completo:

```
SELECT c.numcomanda, lin_com, codfab, codprod  
FROM COMANDA c, LINIA_COMANDA lc  
WHERE c.numcomanda=lc.numcomanda;
```

# SQL: JOINS

## INNER JOINS

- Seleccionarem los campos que sus valores de **claves fòraneas** <-> **claves primarias** coincidan.
- Tendremos que identificar las Tablas seleccionadas.
- Tendremos que relacionar los campos en **ON**.

Se muestran los que tienen datos en las dos tablas

```
SELECT c.numcomanda, lin_com, codfab, codprod  
FROM COMANDA c INNER JOIN LINIA_COMANDA lc  
ON (c.numcomanda=lc.numcomanda);
```

# SQL: JOINS

## LEFT OUTER JOINS

- Seleccionaremos los campos que sus valores de **claves fóraneas**  $\leftrightarrow$  **claves primarias** coincidan Y todo los de la tabla de la izquierda.

Tabla izquierda: se muestra toda  
Tabla derecha: se muestra si coincide

- Esta consulta priorizando los registros de la tabla de la izquierda (la primera mostrada en la consulta) aunque no exista un registro que corresponda a la tabla de la derecha.
- Las columnas de la tabla de la derecha que no tengan correspondencia tendrán valor NULL.

# SQL: JOINS

## LEFT OUTER JOINS

- Tendremos que identificar las Tablas seleccionadas.
- Tendremos que relacionar los campos en **ON**.

```
SELECT c.numcomanda, lin_com, codfab, codprod  
FROM COMANDA c LEFT OUTER JOIN  
LINIA_COMANDA lc  
ON (c.numcomanda=lc.numcomanda);
```

# SQL: JOINS

## RIGHT OUTER JOINS

- Seleccionaremos los campos que sus valores de **claves fòraneas** <-> **claves primarias** coincidan Y todo los de la tabla de la derecha.

Tabla derecha: se muestra toda  
Tabla izquierda: se muestra si coincide

- Esta consulta priorizando los registros de la tabla de la derecha(la segunda mostrada en la consulta) aunque no exista un registro que corresponda a la tabla de la izquierda.
- Las columnas de la tabla de la izquierda que no tengan correspondencia tendrán valor NULL.

# SQL: JOINS

## RIGHT OUTER JOINS

- Tendremos que identificar las Tablas seleccionadas.
- Tendremos que relacionar los campos en **ON**.

```
SELECT c.numcomanda, lin_com, codfab, codprod  
FROM COMANDA c RIGHT OUTER JOIN  
LINIA_COMANDA lc  
ON (c.numcomanda=lc.numcomanda);
```



# SQL: JOINS

## JOINS DE MAS DE DOS TABLAS

Podemos relacionar más de dos tablas si tuvieran claves relacionadas entre ellas:

```
SELECT c.numcomanda, lin_com, codfab, codprod, nom_venedor  
FROM COMANDA c, LINIA_COMANDA lc, VENEDORS v  
WHERE c.numcomanda=lc.numcomanda  
AND  
v.codi =c.codi_venedor;
```

En este caso añadimos una tercera tabla Venedors, recogeremos su nombre.

# SQL: SUBCONSULTAS

## SUBCONSULTAS (SUBQUERIES)

- SELECTS, UPDATES, DELETES admeten subqueries, també conegudes per subconsultas: una consulta dentro de otra consulta.

- Esto significa que cuando tenemos una condición **WHERE** o **HAVING** podemos anidar una **SELECT** en la condición.

Siendo S una consulta SQL válida la podremos tratar de diferentes formas.

# SQL: SUBCONSULTAS

## 1- Si S devuelve un único resultado:

```
SELECT ...  
FROM ...  
WHERE valor= (S)
```

{cualquier operador de comparación: < , > , >= , ...}

```
SELECT oficina, ciudad  
FROM OFICINAS  
WHERE objetivo > (SELECT SUM(e.ventas)  
                   FROM EMPLEADOS  
                   );
```

(Lista las oficinas que el objetivo sea superior a la suma de las ventas de todos los empleados)

# SQL: SUBCONSULTAS

**2. Si S devuelve más de un resultado pero con un solo campo podemos hacer a su vez varias acciones**

**2.1:**

```
SELECT ...  
FROM ...  
WHERE valor IN (S)
```

{Devuelve CIERTO si el valor es igual a alguno de los devueltos}

```
SELECT numemp, nom, oficina  
FROM EMPLEADOS  
WHERE oficina IN (SELECT oficina  
FROM OFICINAS  
WHERE region = 'este');
```

(Lista de los empleados de la oficina del este)

# SQL: SUBCONSULTAS

## 2.2:

```
SELECT ...  
FROM ...  
WHERE valor [>, <, ...] ANY (S)
```

{Devuelve CIERTO si se cumple la condición especificada con  
ALGUNO de los valores que S devuelve}

```
SELECT oficina, ciudad  
FROM OFICINAS  
WHERE objetivo > ANY (SELECT SUM(quota)  
                        FROM EMPLEADOS  
                        GROUP BY oficina);
```

(Lista las oficinas que el objetivo sea superior a alguna de las sumas obtenidas)

# SQL: SUBCONSULTAS

## 2.3:

```
SELECT ...  
FROM ...  
WHERE valor [>, <, ...] ALL (S)
```

{Devuelve CIERTO si se cumple la condición especificada con todos los valores devueltos}

```
SELECT oficina, ciudad  
FROM OFICINAS  
WHERE objetivo > ALL(SELECT SUM(quota)  
FROM EMPLEADOS  
GROUP BY oficina);
```

(Lista las oficinas que el objetivo sea superior a todas las sumas obtenidas)

# SQL: SUBCONSULTAS

## 2.4:

```
SELECT ...  
FROM ...  
WHERE [NOT] EXISTS (S)
```

{Devuelve CIERTO si usamos **EXISTS** y la subconsulta devuelve algun registro o si usamos **NO EXISTS** y la subconsulta no devuelve ningún registro.}

```
SELECT numemp, nom, oficina,  
FROM EMPLEADOS E  
WHERE EXISTS (SELECT e.numemp  
FROM OFICINAS O  
WHERE region='este' AND  
E.Oficina = O.oficina);
```

(Lista los empleados de las oficinas del este.)

# SQL: SUBCONSULTAS

## ***UPDATE con subconsultas:***

```
UPDATE <nom_tabla>  
SET (atr1, ..., atrn) = (SELECT atr1, ..., atr  
                        FROM <nom_taula2>  
                        WHERE <cond>)
```

```
WHERE <condicions>;
```

Actualiza la tabla segun el valor de una seleccion

Ejemplo:

```
UPDATE empleats  
SET (oficina, salari) = (SELECT oficina, salari  
                        FROM empleats  
                        WHERE numemp = 'C2')  
  
WHERE numemp = 'C4';
```



# SQL: SUBCONSULTAS

## ***DELETE con subconsultas:***

```
DELETE FROM <nom_tabla1>  
WHERE campo (NOT) IN (S);  
/* siendo S una subconsulta que puede devolver uno o  
más valores*/
```

Ejemplo:

```
DELETE FROM empleados  
WHERE oficina IN (SELECT oficina  
FROM oficinas  
WHERE ciudad= 'Valencia');
```

# PHPMyAdmin: USUARIOS

- Se pueden generar nuevos usuarios para todas las bases de datos o incluso para algunas bases de datos en específico.
- Se deberán añadir los permisos adecuados: puede ser que a un usuario solo se le permita consultar y a otro solo consultar e insertar,...
- Para ello se debe navegar hasta la pestaña **Permisos** del PHPMyAdmin.

# PHPMyAdmin: USUARIOS

localhost

Bases de datos SQL Estado actual Procesos Privilegios Exportar Importar Variables Juegos de caracteres Más

## Agregar un nuevo usuario

### Información de la cuenta

Nombre de usuario: Use el campo de tex

Servidor: Cualquier servidor  1

Contraseña: Use el campo de tex

Debe volver a escribir:

Generar contraseña:

### Base de datos para el usuario

☒ Ninguna  
☐ Crear base de datos con el mismo nombre y otorgar todos los privilegios  
☐ Otorgar todos los privilegios al nombre que contiene comodín (username\\_%)

### Privilegios globales (Marcar todos / Desmarcar todos)

*Nota: Los nombres de los privilegios de MySQL están expresados en inglés*

Datos	Estructura	Administración	Límites de recursos
<input type="checkbox"/> SELECT <input type="checkbox"/> INSERT <input type="checkbox"/> UPDATE	<input type="checkbox"/> CREATE <input type="checkbox"/> ALTER <input type="checkbox"/> INDEX	<input type="checkbox"/> GRANT <input type="checkbox"/> SUPER <input type="checkbox"/> PROCESS	<i>Nota: si cambia los parámetros de estas opciones a 0 (cero), remueve el límite.</i> MAX. CONECTES PER HOST: <input type="text"/>

# PHPMyAdmin: Modificar datos

- Editando campos con PHPMyAdmin



	id	titulo	texto
  	1	Instalación y uso del phpMyAdmin	Artículo dedicado a la instalación, configuración ...

Con marca:  

- eliminar
- modificar

# PHPMyAdmin: Exportar

- Podremos exportar datos de las tablas o incluso de la base de datos entera seleccionada.
- Los formatos son varios, pero los más comunes son:
  - SQL: genera un fichero con consultas SQL de manera automática con los datos exportados.
  - CSV: fichero con los datos separados por comas, muy útil si queremos trabajar con Microsoft Excel.

# PHPMyAdmin: Importar

- Podremos importar datos a las tablas o incluso de bases de datos enteras, siempre con un limite impuesto por el PhpMyAdmin.
- Para modificar este limite se tiene que modificar el fichero php.ini como si quisiéramos modificar el tamaño máximo de la subida de archivos:

- upload\_max\_filesize** (Default 2M)
- memory\_limit** (Default 16M) (En caso de tiempo de procesar un archivo tan grande)
- post\_max\_size** (Default 8M)

Bajar en formato CSV de Excel

Importar:

- Borrar linea con los nombres de campos
- Poner el Id correlativo que toque
- Cambiar la fecha, si hay, a formato ingles
- Al cargar cambiar "," a "·"
- Hacer CLICK en "Actualizar datos cuando las llaves importadas están duplicadas"

- Los formatos que se aceptan son también muy parecidos a la exportación y los más usados siguen siendo ficheros SQL con instrucciones sql en ellos o CSV con datos separados por coma.