

# Bases de Datos

1º DAW

## UD5.1 - Modificación Datos SQL



# CONTENIDOS

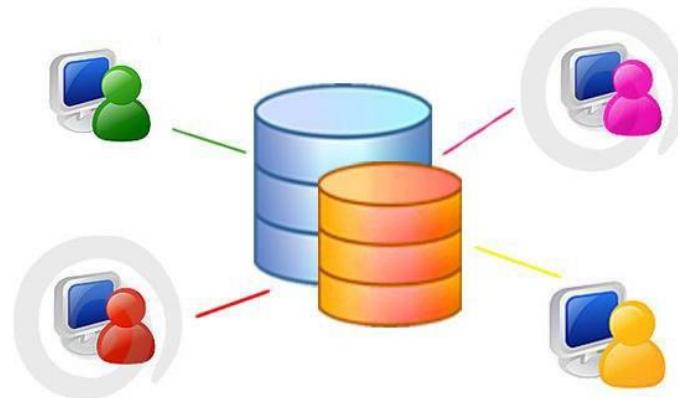
1. Introducción
2. Inserción (INSERT)
3. Modificación (UPDATE)
4. Borrado (DELETE)



# INTRODUCCIÓN

## Modificación de datos

En esta UD vamos a tratar el de la actualización de esos datos, es decir insertar nuevas filas, borrar filas o cambiar el contenido de las filas de una tabla. Estas operaciones modifican los datos almacenados en las tablas pero no su estructura, ni su definición.



Empezaremos por ver cómo insertar nuevas filas (con la sentencia **INSERT INTO**), veremos una variante (la sentencia **SELECT... INTO**), después veremos cómo borrar filas de una tabla (con la sentencia **DELETE**) y por último cómo modificar el contenido de las filas de una tabla (con la sentencia **UPDATE**).

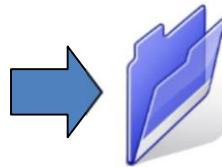
## INSERCIÓN

Datos nuevos



## INSERT INTO...VALUES

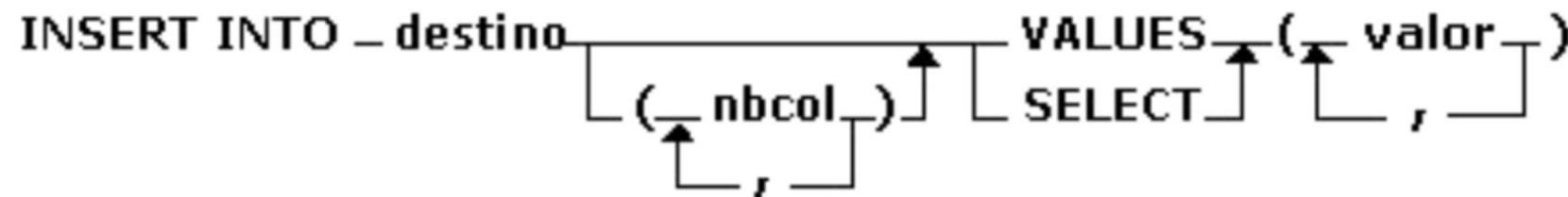
Insertar una fila



La inserción de nuevos datos en una tabla se realiza añadiendo filas enteras a la tabla, la sentencia SQL que lo permite es la orden INSERT INTO.

La inserción se puede realizar de una fila o de varias filas de golpe, veremos las dos opciones por separado y empezaremos por la inserción de una fila.

La sintaxis es la siguiente:

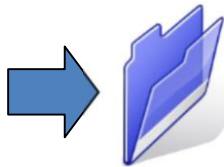


Esta sintaxis se utiliza para insertar una sola fila cuyos valores indicamos después de la palabra reservada VALUES.

En castellano la sentencia se leería: INSERTA EN destino...VALORES ....

## INSERT INTO...VALUES

Detalles 1/2



Los registros se agregan siempre al final de la tabla.

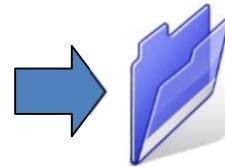
*Destino* es el nombre de la tabla donde vamos a insertar la fila, también se puede utilizar un nombre de consulta, consulta que tenga como origen de datos una única tabla.

La palabra reservada VALUES se puede sustituir por la palabra SELECT.

A continuación de la palabra VALUES, entre paréntesis se escriben los valores que queremos añadir. Estos valores se tienen que escribir de acuerdo al tipo de dato de la columna donde se van a insertar (encerrados entre comillas simples '' para valores de tipo texto, por ejemplo) la asignación de valores se realiza por posición, el primer valor lo asigna a la primera columna, el segundo valor a la segunda columna, así sucesivamente...

## INSERT INTO...VALUES

Detalles 2/2



Cuando la tabla tiene una columna de tipo contador (AutoNumber), lo normal es no asignar valor a esa columna para que el sistema le asigne el valor que le toque según el contador, si por el contrario queremos que la columna tenga un valor concreto, lo indicamos en la lista de valores.

Cuando no se indica ninguna lista de columnas después del destino, se asume por defecto todas las columnas de la tabla, en este caso, los valores se tienen que especificar en el mismo orden en que aparecen las columnas en la ventana de diseño de dicha tabla, y se tiene que utilizar el valor NULL para llenar las columnas de las cuales no tenemos valores.

Cuando se dan valores a todas las columnas de la tabla:

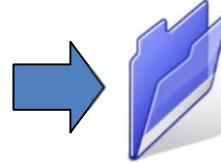
**Insert Into** <tabla>  
**Values** (*valor<sub>1</sub>*, *valor<sub>2</sub>*, *valor<sub>3</sub>*, ..., *valor<sub>N</sub>*)

Si hay alguno que no recibe valor, hay que utilizar:

**Insert Into** <tabla> (*col<sub>1</sub>*, *col<sub>2</sub>*, ..., *col<sub>N</sub>*)  
**Values** (*valor<sub>1</sub>*, *valor<sub>2</sub>*, ..., *valor<sub>N</sub>*)

## INSERT INTO...VALUES

Ejemplo 1 – Todos los campos



Se tiene la siguiente tabla:

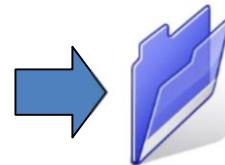
```
create table empleado (
    id int primary key,
    apellido varchar(20) not
    null,
    oficio varchar(20),
    director int not null,
    fecha_alta date,
    salario int,
    comision int,
    departamento varchar (20)
)
```

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with icons for connection, schema browser, and other database management functions. Below the toolbar, a status bar displays the current connection information. The main area consists of two panes: a left pane titled 'columns 1 x' and a right pane titled 'describe empleado'. The left pane contains a table with columns: Grilla, Texto, and Record. The right pane displays the results of the 'describe empleado' query, which lists the table's columns with their respective details.

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	[NULL]	
apellido	varchar(20)	NO		[NULL]	
oficio	varchar(20)	YES		[NULL]	
director	int	NO		[NULL]	
fecha_alta	date	YES		[NULL]	
salario	int	YES		[NULL]	
comision	int	YES		[NULL]	
departamen	varchar(20)	YES		[NULL]	

## INSERT INTO...VALUES

Ejemplo 1 – Todos los campos



La manera en que se pueden hacer las inserciones de datos en una tabla no es única, existen diferentes posibilidades y alternativas:

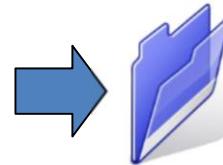
- `INSERT INTO n_table (col1, ..., coln) VALUES (val1, ... valn)`
- Se pueden indicar el nombre de las columnas desordenadas y luego los valores deben de coincidir según el orden especificado en la declaración de los valores:

```
INSERT INTO n_tabla(col1, col3, col5, col2, col4) VALUES  
          (val1, val3, val5, val2, val4)
```

- Si hay alguna columna que no se quiere incluir valor se debe de indicar el orden de inserción: `INSERT INTO n_tabla (col1, col3) VALUES (val1, val3)`. La columna que no se indique tendrá por valor `NULL` o el que se le haya indicado que deba de tener por defecto
- Se puede obviar el paso de indicar las columnas a insertar y, en este caso la inserción se hace en el orden en el que estén las columnas y no puede dejarse ninguna columna: `INSERT INTO n_table VALUES (val1, ..., valn)`

## INSERT INTO...VALUES

Ejemplo 1 – Todos los campos



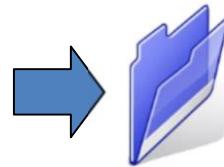
```
insert into empleado values (1, 'garcia', 'contable', 2, '2023-01-09',
1000, 1, 'contabilidad')
insert into empleado values (2, 'garcia', 'director', 2, '2023-01-09',
1000, 1, 'contabilidad')
insert into empleado values (3, 'garcia', null, 2, '2023-01-09', 1000,
1, 'contabilidad')
insert into empleado (id, apellido, director, fecha_alta, salario,
departamento)
values (4, 'garcia', 2, '2023-01-09', 1000, 'contabilidad')
```

```
insert into empleado (id, apellido, director, fecha_alta, salario, departamento)

values (4, 'garcia', 2, '2023-01-09', 1000, 'contabilidad')
```

## INSERT INTO...VALUES

¿Cuál es mejor?



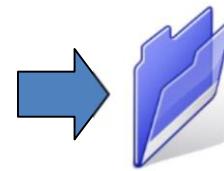
Observar que ahora hemos variado el orden de los valores y los nombres de columna no siguen el mismo orden que en la tabla origen, no importa, lo importante es poner los valores en el mismo orden que las columnas que enunciamos.

El utilizar la opción de poner una **lista de columnas** podría parecer peor ya que se tiene que escribir más pero realmente tiene ventajas sobre todo cuando la sentencia la vamos a almacenar y reutilizar:

1. La sentencia queda más fácil de interpretar leyéndola vemos qué valor asignamos a qué columna.
2. Nos aseguramos que el valor lo asignamos a la columna que queremos.
3. Si por lo que sea cambia el orden de las columnas en la tabla en el diseño, no pasaría nada mientras que de la otra forma intentaría asignar los valores a otra columna, esto produciría errores de 'tipo no corresponde' y lo que es peor podría asignar valores erróneos sin que nos demos cuenta.
4. Si se añade una nueva columna a la tabla en el diseño, la primera sentencia INSERT daría error ya que el número de valores no corresponde con el número de columnas de la tabla, mientras que la segunda INSERT no daría error y en la nueva columna se insertaría el valor predeterminado.

## INSERT INTO...VALUES

### Errores



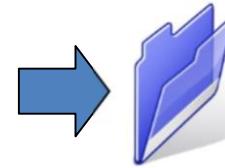
Errores que se pueden producir cuando se ejecuta la sentencia INSERT INTO:

- Si la tabla de destino tiene clave principal y en ese campo intentamos no asignar valor, asignar el valor nulo o un valor que ya existe en la tabla, el motor de base de datos no añade la fila y da un mensaje de error.
- Si tenemos definido un índice único (sin duplicados) e intentamos asignar un valor que ya existe en la tabla también devuelve el mismo error.
- Si la tabla está relacionada con otra, se seguirán las reglas de integridad referencial.



## INSERT INTO...SELECT

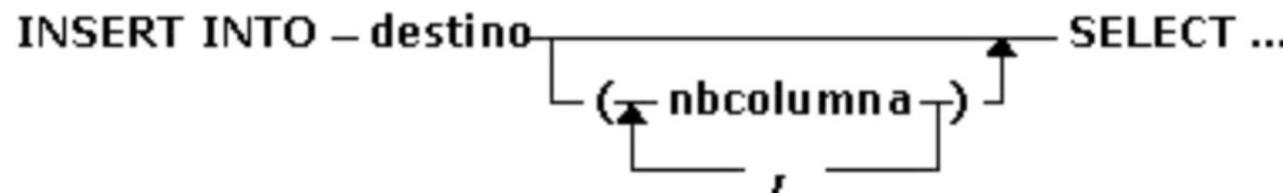
Insertar varias filas



Podemos insertar en una tabla varias filas con una sola sentencia SELECT INTO si los valores a insertar se pueden obtener como resultado de una consulta. En este caso sustituimos la cláusula *VALUES lista de valores* por una sentencia SELECT

Cada fila resultado de la SELECT forma una lista de valores que son los que se insertan en una nueva fila de la tabla destino. Es como si tuviésemos una INSERT...VALUES por cada fila resultado de la sentencia SELECT.

La sintaxis es la siguiente:



## INSERT INTO...SELECT

### Detalles 1/2

El origen de la SELECT puede ser el nombre de una consulta guardada, un nombre de tabla o una composición de varias tablas.

Cada fila devuelta por la SELECT actúa como la lista de valores que vimos con la INSERT...VALUES por lo que tiene las mismas restricciones en cuanto a tipo de dato, etc. La asignación de valores se realiza por posición por lo que la SELECT debe devolver el mismo número de columnas que las de la tabla destino y en el mismo orden, o el mismo número de columnas que indicamos en la lista de columnas después de destino.

Las columnas de la SELECT no tienen porque llamarse igual que en la tabla destino ya que el sistema sólo se fija en los valores devueltos por la SELECT.

Si no queremos asignar valores a todas las columnas entonces tenemos que indicar entre paréntesis la lista de columnas a llenar después del nombre del destino.

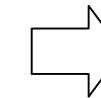
## INSERT INTO...SELECT

### Detalles 2/2

El estándar ANSI/ISO especifica varias restricciones sobre la consulta que aparece dentro de la sentencia INSERT:

- La consulta no puede tener una cláusula ORDER BY.
- La tabla destino de la sentencia INSERT no puede aparecer en la cláusula FROM de la consulta o de ninguna subconsulta que ésta tenga. Esto prohíbe insertar parte de una tabla en sí misma.
- La consulta no puede ser la UNION de varias sentencias SELECT diferentes.
- El resultado de la consulta debe contener el mismo número de columnas que las indicadas para insertar y los tipos de datos deben ser compatibles columna a columna.

# INSERT INTO...SELECT



## Ejemplo

Sean las dos siguientes tablas ALUMNOS y ALUM0405 respectivamente:

	dni	nombre	apellido	direccion	provincia	telefono
1	123456789	alumno1	ape1	calle mayor	madrid	123.456.987
2	123456798	alumno2	ape12	calle arenal	madrid	123.456.798

```
insert into alumno2 select dni, nombre, apellido,  
concat(direccion, concat(' de ', provincia)),  
telefono  
from alumno
```

The screenshot shows the MySQL Workbench interface with two panes. The top pane contains the SQL query:

```
49 select * from alumno2  
50  
51  
52
```

The bottom pane shows the results of the query in the 'alumno2' table:

	dni	nombre	apellido	direccion	telefono
1	123456789	alumno1	ape1	calle mayor de madrid	123.456.987
2	123456798	alumno2	ape12	calle arenal de madrid	123.456.798

A tooltip labeled 'Valor' is shown next to the value '123456789' in the telefono column of the first row.

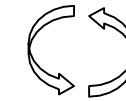
## MODIFICAR

Cambiar datos



## UPDATE

Modificar el contenido de las filas



La sentencia **UPDATE** modifica los valores de una o más columnas de una tabla en las filas seleccionadas.

La sintaxis es la siguiente:

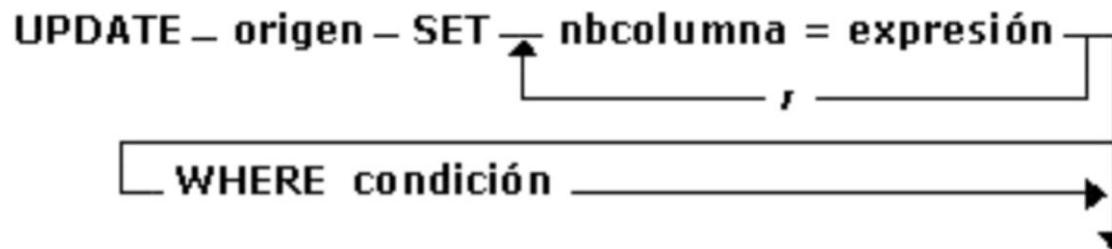
```
Update <tabla>
Set
    col1 = valor1,
    col2 = valor2,
    ...
    colN = valorN
[ Where <condición> ]
```

# UPDATE

## Detalles



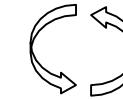
Expliquemos la sintaxis:



- *Origen* puede ser un nombre de tabla, un nombre de consulta o una composición de tablas.
- La cláusula *SET* especifica qué columnas van a modificarse y qué valores asignar a esas columnas.
- *nbcolumna*, es el nombre de la columna a la cual queremos asignar un nuevo valor por lo tanto debe ser una columna de la tabla origen. El SQL estándar exige nombres sin cualificar pero algunas implementaciones sí lo permiten.
- La *expresión* en cada asignación debe generar un valor del tipo de dato apropiado para la columna indicada. La expresión debe ser calculable a partir de los valores de la fila que se está actualizando. También se puede utilizar una subconsulta que de un único valor.

## UPDATE

### Ejemplo



Vamos a actualizar el teléfono de los datos que hemos insertado recientemente:

```
update alumno set telefono=123654998 where dni =
'123456789'
select * from alumno
```

The screenshot shows a MySQL command-line interface. The command window contains the following SQL code:

```
24 update alumno set telefono=123654998 where dni = '123456789'
25 select * from alumno
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
```

Below the command window is a results table titled "alumno 1". The table has columns: dni, nombre, apellido, direccion, provincia, and telefono. The data is as follows:

	dni	nombre	apellido	direccion	provincia	telefono
1	123456789	alumno1	ape1	calle mayor	madrid	123.654.998
2	123456798	alumno2	ape12	calle arenal	madrid	123.456.798

## UPDATE

### Ejemplo - Explicación



La cláusula WHERE indica qué filas van a ser modificadas. Si se omite la cláusula WHERE se actualizan todas las filas.

En la condición del WHERE se puede incluir una subconsulta. En SQL standard la tabla que aparece en la FROM de la subconsulta no puede ser la misma que la tabla que aparece como origen.

Si para el cálculo de expresión se utiliza una columna que también se modifica, el valor que se utiliza es el antes de la modificación, lo mismo para la condición de búsqueda.

Cuando se ejecuta una sentencia UPDATE primero se genera el origen y se seleccionan las filas según la cláusula WHERE. A continuación se coge una fila de la selección y se le aplica la cláusula SET, se actualizan todas las columnas incluidas en la cláusula SET a la vez por lo que los nombres de columna pueden especificarse en cualquier orden. Después se coge la siguiente fila de la selección y se le aplica del mismo modo la cláusula SET, así sucesivamente con todas las filas de la selección.

# BORRAR

Eliminar datos



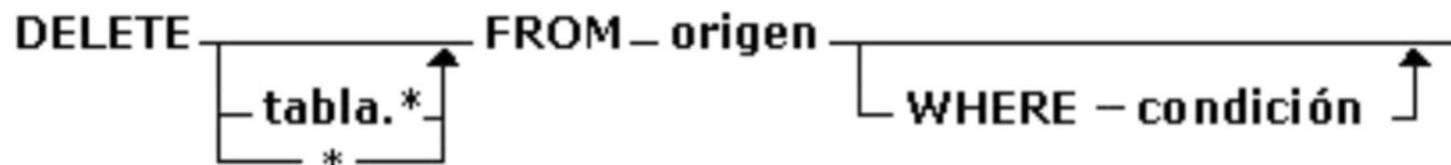
## DELETE

Eliminar filas

La sentencia DELETE elimina filas de una tabla.

La sintaxis es la siguiente:

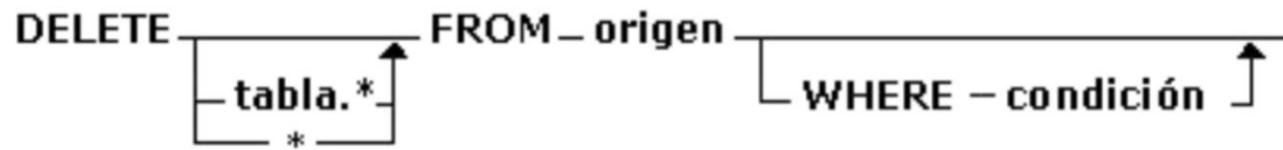
**Delete From <tabla>  
[ Where <condición> ]**



Origen es el nombre de la tabla de donde vamos a borrar.

## DELETE

### Detalles



La cláusula WHERE sirve para especificar qué filas queremos borrar. Se eliminaran de la tabla todas las filas que cumplan la condición. Si no se indica la cláusula WHERE, se borran **TODAS** las filas de la tabla.

En la condición de búsqueda de la sentencia DELETE, se puede utilizar una subconsulta. En SQL standard la tabla que aparece en la FROM de la subconsulta no puede ser la misma que la tabla que aparece en la FROM de la DELETE.

Si la tabla donde borramos está relacionada con otras tablas se podrán borrar o no los registros siguiendo las reglas de integridad referencial definidas en las relaciones.

**Atención:** Una vez borrados, los registros no se pueden recuperar.

## DELETE

### Ejemplo

Antes de hacer un DELETE se recomienda hacer siempre antes una sentencia SELECT para comprobar los valores que se van a borrar.

```
26
27
28 select * from alumno
29
30 delete from alumno where apellido = 'ape12'
31
32 select * from alumno
```

alumno 1 ×

select \* from alumno | *Enter a SQL expression to filter results (use Ctrl+Space)*

	dni	nombre	apellido	direccion	provincia	telefono	Valor
1	123456789	alumno1	ape1	calle mayor	madrid	123.654.998	123456789

Renovar Save Cancel Exportar datos ... 200 1

### RECORDATORIO:

- DROP. Eliminamos tablas de una base de datos
- TRUNCATE. Vaciamos tablas de una base de datos (OJO: no borra la tabla, tan sólo la deja sin datos)
- DELETE. Borramos filas de una tabla con la condición que se indique