
	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

## Tema 7

# Manipulación de datos

### Índice

1.	Introducción .....	1
2.	Inserción de datos. Orden INSERT y orden REPLACE .....	1
2.1.	Inserción con SELECT .....	2
2.2.	Replace .....	2
3.	Modificación de datos. Orden UPDATE .....	3
3.1.	Update con select .....	3
3.2.	Update de multiples tablas .....	4
4.	Eliminación de datos. Orden DELETE .....	5
5.	Vaciado de tablas .....	5
6.	Importar y exportar datos en MariaDB/MySQL .....	5
6.1.	Importar a partir de ficheros externos .....	5
6.2.	Exportar a ficheros .....	7

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

## 1. Introducción

Hasta ahora nos hemos dedicado a consultar datos de la base de datos mediante la sentencia SELECT. En este tema veremos cómo cambiar los datos de las tablas de la base de datos; aprenderemos a insertar nuevas filas en una tabla, a actualizar los valores de las columnas en las filas y a borrar filas.

## 2. Inserción de datos. Orden INSERT y orden REPLACE

Mediante la orden INSERT podremos añadir filas de datos as una tabla.

El formato de esta orden es el siguiente:

```
INSERT INTO NombreTabla [(columna [, columna ] ...)] VALUES (valor [, valor ] ...);
```

Donde:

- NombreTabla es la tabla en la que se van a insertar las filas.
- [(columna [, columna ] . . . )] representa la columna o columnas donde se van a introducir valores. Si las columnas no se especifican en la cláusula INSERT, se consideran, por defecto, todas las columnas de la tabla.
- (valor [, valor] . . . ) representa los valores que se van a dar a las columnas. Estos se deben corresponder con cada una de las columnas que aparecen; además, deben coincidir con el tipo de dato definido para cada columna.

Si no se da la lista de columnas, se han de introducir valores en todas las columnas en el mismo orden en que estas estén definidas en la tabla como se muestra a continuación:

```
INSERT INTO escritores (nombre, dirección, población) VALUES ('Neal Stephenson', 'c/ Criptonomicón','Vigo');
INSERT INTO escritores VALUES (NULL, DEFAULT, 'Neal Stephenson', 'c/ Criptonomicón', NULL, '5559999999');
```

Observamos en la sentencia del ejemplo anterior que:

- Las columnas a las que damos valores se identifican por su nombre.
- La asociación columna-valor es posicional.
- Los valores que se dan a las columnas deben coincidir con el tipo de dato definido en la columna.
- Los valores constantes de tipo carácter han de ir encerrados entre comillas.
- Podemos usar el valor null y el valor default que insertaran respectivamente el valor nulo o el valor por defecto que la columna tenga definido.
- Se puede escoger que columnas rellenar, aunque siempre teniendo en cuenta que solo se pueden ignorar las que admiten un valor null, las que son autoincrementadas o las que tienen definido un valor por defecto. Si no se especifica que columnas se están rellenando, hay que dar valores para todas las columnas.

También se pueden insertar varias filas a la vez separándolas por comas:

```
INSERT INTO escritores (nombre, dirección, población) VALUES ('Neal Stephenson', 'c/ cripto', 'Vigo'),
('Frank Herbert', 'c/ dune 10191', null),
('Larry Niven', 'c/ anillo 2', default);
```

Existe otra sintaxis alternativa, que consiste en indicar el valor para cada columna:

```
INSERT INTO NombreTabla SET columna1=valor1, columna2=valor2, ...;
INSERT INTO escritores SET nombre="R. Daneel Olivaw", población ="Aurora";
```

COLEXIO <b>VIVAS</b> S.L.	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Para que una fila se considere duplicada debe tener el mismo valor que una fila existente para una clave principal o para una clave única. En tablas en las que no exista clave primaria ni índices de clave única no tiene sentido hablar de filas duplicadas. Es más, en esas tablas es perfectamente posible que existan filas con los mismos valores para todas las columnas.

Si intentamos insertar dos filas con el mismo valor de clave primaria o campo único se producirá un error y no se ejecutará la sentencia. Existe una opción que podemos utilizar cuando nos encontramos con claves duplicadas: le indicamos que queremos modificar el contenido, es decir, realizar un reemplazo del registro. Para eso utilizamos las palabras reservadas ON DUPLICATE KEY UPDATE.

#### EJEMPLO

Imaginemos un contador de visitas, que contabiliza el número de visitas de una dirección IP a un sitio web así como la hora del último acceso: Visitas (IP, cantidad, fechaUltima)

```
INSERT INTO Visitas VALUES (inet_aton('192.168.0.1'), 1, now())
on DUPLICATE KEY UPDATE cantidad=cantidad +1, fechaUltima=now();
```

## 2.1. Inserción con SELECT

Hasta el momento sólo hemos insertado una fila, pero si añadimos a INSERT una consulta, es decir, una sentencia SELECT, se añaden tantas filas como devuelva la consulta. El formato de INSERT con SELECT es el siguiente:

```
INSERT INTO NombreTabla1 [(columna [, columna]
SELECT {columna [, columna] ... | *}
FROM NombreTabla2 .....;
```

#### EJEMPLO

Disponemos además de la tabla escritores de la tabla bestSellers cuyos atributos son idénticos. Queremos insertar en la tabla bestSellers los datos de los escritores con un id menor que 5:

```
INSERT INTO bestSellers (dni, nombre, direc, pobla, telef)
SELECT dni, nombre, direc, pobla, telef FROM escritores WHERE id<5;
```

Ya que las dos tablas tienen los mismos atributos la sentencia anterior sería equivalente a:

```
INSERT INTO bestSellers SELECT * FROM escritores WHERE id<5;
```

## 2.2. Replace


La sentencia REPLACE, es una alternativa para INSERT, que sólo se diferencia en que si existe algún registro anterior con el mismo valor para una clave primaria o única, se elimina el viejo y se inserta el nuevo en su lugar, sino simplemente se inserta.

Las mismas sintaxis que existen para INSERT, están disponibles para REPLACE:

```
REPLACE [INTO] NombreTabla [(columna [, columna] ... ) ] VALUES (valor [, valor ] ...);
```

Ejemplos:

```
REPLACE INTO escritores VALUES (15, '9999998', 'R. Giskard Reventlov', 'c/ Solaria', 'Aurora', NULL);
REPLACE escritores (nombre, direc, pobla) VALUES ('R. Daneel Olivaw', 'c/ Solaria', 'Aurora');
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

### 3. Modificación de datos. Orden UPDATE

Para actualizar los valores de las columnas de una o varias filas de una tabla utilizamos la orden UPDATE, cuyo formato es el siguiente:

```
UPDATE NombreTabla
SET columna1=valor1, ..., columnaN=valorN
WHERE condición;
```

Donde

- NombreTabla es la tabla cuyas columnas se van a actualizar.
- SET indica las columnas que se van a actualizar y sus valores.
- WHERE selecciona las filas que se van a actualizar. Si se omite, la actualización afectará a todas las filas de la tabla.

#### EJEMPLO

Queremos actualizar el nombre del escritor con id 8 a Elijah Baley y su población a Tierra.

```
update escritores set nombre="Elijah Baley", población="Tierra" where id=8;
```

Si en el caso anterior hubiésemos omitido la cláusula where se modificarían los campos nombre y población de todas la filas de la tabla.

Se puede utilizar además los valores NULL y DEFAULT como nuevo valor de una columna, los cuales establecer el valor a nulo y al valor por defecto respectivamente:

```
UPDATE escritores SET nombre=NULL WHERE id=8;
UPDATE escritores SET nombre=DEFAULT WHERE id=8;
```

Si la expresión indicada en el WHERE se cumple para varias filas mediante LIMIT se puede limitar el número de registros que se actualizan.

```
UPDATE escritores SET nombre=NULL WHERE nombre like "%a%" LIMIT 2;
```

En este caso se actualizarían los dos primeros escritores que posean una a en su nombre.

#### 3.1. Update con select

Podemos incluir una subconsulta en una sentencia UPDATE que puede estar contenida en la cláusula where o puede formar parte de SET. Cuando la subconsulta forma parte de SET, debe seleccionar una única fila y el mismo número de columnas (con formatos de datos adecuados) que las que hay entre paréntesis al lado de SET.

Los formatos son:

```
UPDATE <NombreTabla>
SET (columna1, columna2, ...) = (SELECT col1, col2, ...) WHERE condición;
```

```
UPDATE <NombreTabla>
SET columna1=valor1, columna2=valor2, ... WHERE columna3=(SELECT ...);
```

```
UPDATE <NombreTabla>
SET columna1=(SELECT col1 ...), columna2=(SELECT col2 ...) WHERE condición;
```

<div> <div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div> </div>	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

#### EJEMPLOS

Pon en mayúscula la población de aquellos que tengan un id mayor que el bestseller Frederik Pohl.

```
UPDATE escritores SET nombre=UPPER(nombre) WHERE id>(SELECT id FROM bestsellers WHERE nombre='Frederik Pohl');
```

Actualizar los teléfonos de los escritores con id < 4 al teléfono del bestseller que tenga id 2:

```
UPDATE escritores SET telef=(SELECT telef FROM bestSellers WHERE id=2) WHERE id<4;
```

### 3.2. Update de multiples tablas<sup>1</sup>

En MariaDB/MySQL se pueden actualizar simultaneamente varias tablas especificandolas entre la clausula update y set.

Tenemos las siguientes tablas:

Tabla 1

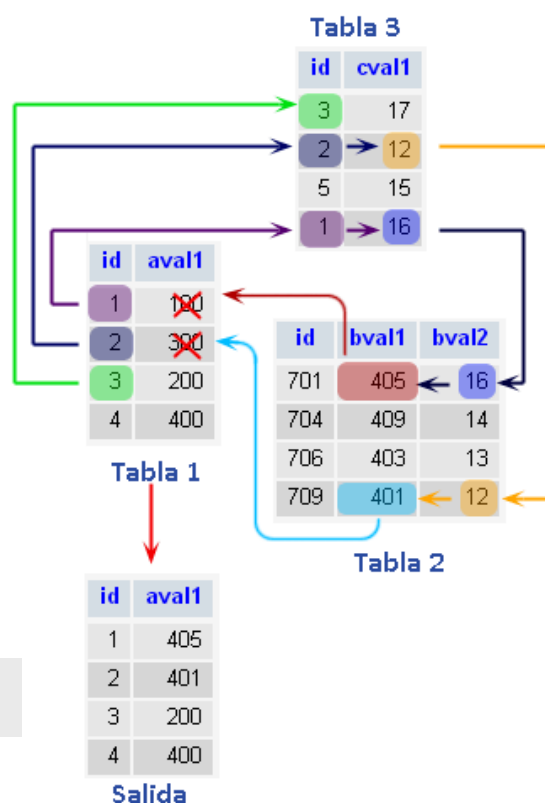
id	aval1
1	100
2	300
3	200
4	300

Tabla 2

id	bval1	Bval2
701	405	16
704	409	14
706	403	13
709	401	12

Tabla 3


id	cval1
3	17
2	12
5	15
1	18



Podemos usar la sentencia:

```
UPDATE tabla1, tabla2, tabla3 SET tabla1.aval1 = tabla2.bval1
WHERE tabla1.id = tabla3.id AND tabla2.bval2 = tabla3.cval1 ;
```

<sup>1</sup> <http://www.w3resource.com/mysql/update-table/update-table.php>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

## 4. Eliminación de datos. Orden DELETE

Para eliminar una fila o varias filas de una tabla se usa la orden DELETE. Esta instrucción es parecida a la instrucción SELECT, con la salvedad de que como se elimina el registro completo no es necesario especificar ninguna columna. La cláusula WHERE es esencial para eliminar sólo aquellas filas deseadas (sin la cláusula WHERE, DELETE borrará todas las filas de la tabla). La condición puede incluir una subconsulta.

Éste es su formato:

```
DELETE FROM NombreTabla WHERE Condición;
```

### EJEMPLO

Queremos borrar el escritor con id 4 dentro de la tabla escritores:

```
DELETE FROM escritores WHERE id=4;
```

Queremos borrar todas las filas de la tabla escritores:

```
DELETE FROM escritores;
```

## 5. Vaciado de tablas

Quando queremos eliminar todas las filas de una tabla, vimos en el punto anterior que podíamos usar una sentencia delete sin condiciones. Sin embargo, existe la sentencia alternativa, truncate<sup>2</sup>, que realiza la misma tarea de una forma mucho más rápida y eficiente.

La diferencia es que delete hace un borrado de la tabla fila a fila mientras que truncate borra la tabla y la vuelve a crear vacía, lo que es mucho más eficiente.

El formato de la orden truncate es:

```
TRUNCATE TABLE nombretabla;
```

## 6. Importar y exportar datos en MariaDB/MySQL

MariaDB/MySQL permite exportar tablas a diferentes formatos de texto, así como importar datos a partir de ficheros de textos en diversos formatos.


Esto se puede usar para exportar los datos de nuestras bases de datos a otras aplicaciones, o bien para importar datos desde otras fuentes a nuestras tablas. También se puede usar para hacer copias de seguridad y restaurarlas posteriormente.

### 6.1. Importar a partir de ficheros externos

Para leer el contenido de un fichero de texto e insertarlo en una tabla disponemos de la sentencia LOAD DATA, cuya sintaxis es:

```
LOAD DATA [ LOCAL ] INFILE 'ruta+nombreFichero.txt'
[ REPLACE | IGNORE ] INTO TABLE nombreTabla
[ FIELDS
  [ TERMINATED BY '\t' ]
  [ [ OPTIONALLY ] ENCLOSED BY '"' ]
  [ ESCAPED BY '\\' ]
]
```


<sup>2</sup> <https://mariadb.com/kb/en/mariadb/truncate-table>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

```
[ LINES
  [ STARTING BY " ]
  [ TERMINATED BY '\n' ]
]
[ IGNORE número LINES ]
[ (nombreColumna,...) ]
```

Donde:

- La cláusula LOCAL indica, si aparece, que el fichero está en el ordenador del cliente. Si no se especifica el fichero de texto se buscará en el servidor que ejecuta MariaDB/MySQL, concretamente en el mismo directorio donde esté la base de datos. Esto nos permite importar datos desde nuestro ordenador en un sistema en que el servidor de MariaDB/MySQL se encuentra en otra máquina.
- Las cláusulas REPLACE e IGNORE afectan al modo en que se tratan las filas leídas que contengan el mismo valor para una clave principal o única para una fila existente en la tabla. Si se especifica REPLACE se sustituirá la fila actual por la leída. Si se especifica IGNORE el valor leído será ignorado.
- La parte INTO TABLE nombreTabla indica en qué tabla se insertarán los valores leídos.
- La cláusula FIELDS se refiere a las opciones de cada columna:
  - TERMINATED BY 'carácter': nos permite elegir el carácter delimitador que se usa para separar cada columna. Por defecto, el valor que se usa es el tabulador, pero podemos usar ';;', etc.
  - [OPTIONALLY] ENCLOSED BY 'carácter': sirve para elegir el carácter usado para entrecomillar cada columna. Por defecto se considera que las columnas no van entrecomilladas, pero se puede elegir cualquier carácter. Si se añade la palabra OPTIONALLY indica que solo estarán entrecomilladas las columnas de texto y fecha.
  - ESCAPED BY 'carácter': sirve para indicar el carácter que se usa para escapar aquellos caracteres que pueden dificultar la lectura posterior del fichero. Por ejemplo, si terminamos las columnas con ',' y no las entrecomillamos, un carácter ',' dentro de una columna de texto se interpretará como un separador de columnas. Para evitar esto se puede escapar esa coma con otro carácter. Por defecto se usa el carácter '\'.
- La cláusula LINES se refiere a las opciones para cada fila:
  - STARTING BY 'carácter': permite seleccionar el carácter con el que comienza cada línea. Por defecto no se usa ningún carácter para ello.
  - TERMINATED BY 'carácter': permite elegir el carácter con el que termina cada línea. Por defecto es el retorno de línea, pero se puede usar cualquier otro carácter o caracteres, por ejemplo '\r\n'.
- La cláusula IGNORE número LINES, nos permite que las primeras 'número' de líneas no se importen. Es frecuente que los ficheros de texto que usaremos como fuente de datos contengan algunas cabeceras que expliquen el contenido del fichero, o que contengan los nombres de cada columna. Usando esta cláusula podemos ignorarlas.
- La última parte nos permite indicar las columnas a la que serán asignadas cada una de las columnas leídas, esto será útil si el orden de las columnas en la tabla no es el mismo que en el fichero de texto, o si el número de columnas es diferente.

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Supongamos que queremos añadir el contenido del fichero siguiente a la tabla "clientes":

Fichero de datos de "clientes"

fecha,nombre

2009-03-15,Carlos

2011-09-09,\N

1910-04-15,Santiago

Como vemos, hay dos filas al principio que no contienen datos válidos, las columnas están separadas con comas y, en el fichero las líneas terminan con "\r\n". Usaremos el símbolo \N en el fichero origen de datos cuando un campo tiene un valor null (no contiene datos).

La sentencia adecuada para leer los datos es:

```
LOAD DATA LOCAL INFILE "/home/clientes.txt"
INTO TABLE clientes
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\r\n'
IGNORE 2 LINES
(fecha,nombre);
```

Con lo que el contenido del fichero clientes.txt se añadirá a la tabla clientes.

## 6.2. Exportar a ficheros

Para extraer datos desde una base de datos a un fichero se usa la sentencia:

```
SELECT ... INTO OUTFILE
```

El resto de las cláusulas de SELECT siguen siendo aplicables, la única diferencia es que la salida de la selección se envía a un fichero en lugar de hacerlo a la consola.

La sintaxis de la parte INTO OUTFILE es:

```
[ INTO OUTFILE 'file_name' export_options ]
```

Donde file\_name es el nombre del fichero de salida. Ese fichero no debe existir, ya que en caso contrario la sentencia fallará.

Las opciones de exportación son las mismas que para las cláusulas FIELDS y LINES de LOAD DATA las cuales nos permiten crear diferentes formatos de ficheros de salida.

Por ejemplo, para obtener un fichero de texto a partir de la tabla 'gente', con las columnas delimitadas por ';', entrecomillando las columnas de texto con " y separando cada fila por la secuencia '\n', usaremos la siguiente sentencia:

```
SELECT * INTO OUTFILE "d:\gente2.txt"
FIELDS TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM gente;
```