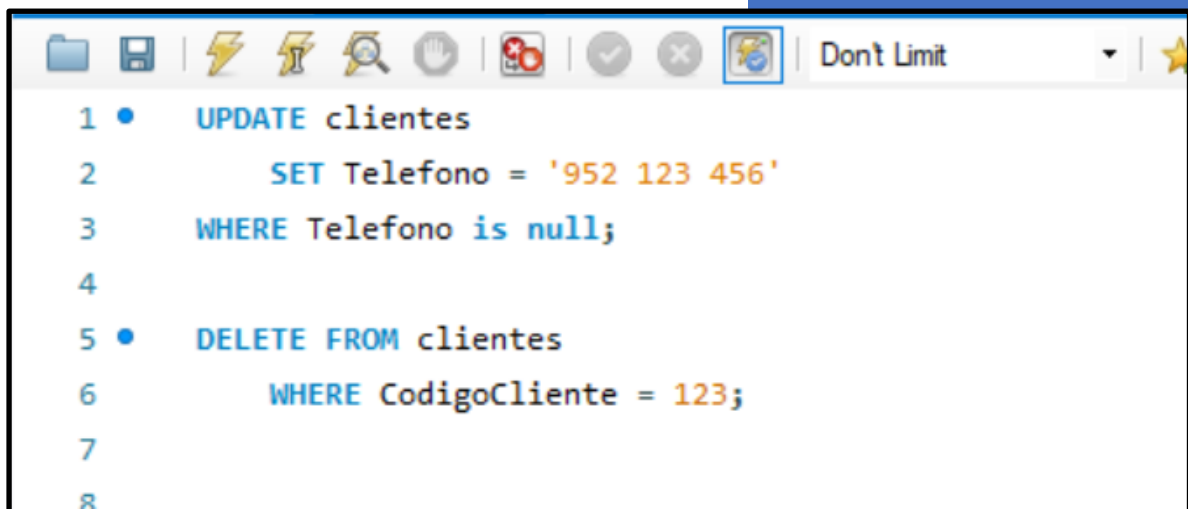


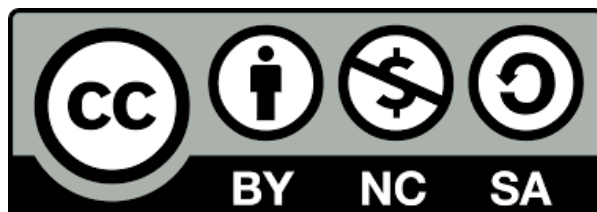
# Tema 5

## Otros usos de SELECT Actualización y borrado de datos



```
1 • UPDATE clientes
2     SET Telefono = '952 123 456'
3     WHERE Telefono is null;
4
5 • DELETE FROM clientes
6     WHERECodigoCliente = 123;
7
8
```

Este documento ha sido elaborado para el alumnado del módulo “Bases de datos” de los CFGS DAM y DAW del IES Playamar.



Más información en <https://creativecommons.org/licenses/by-nc-sa/3.0/es/>

## Contenido

|           |                                    |           |
|-----------|------------------------------------|-----------|
| <b>1.</b> | <b>Introducción .....</b>          | <b>1</b>  |
| <b>2.</b> | <b>Otros usos de SELECT .....</b>  | <b>2</b>  |
| 2.1.      | Funciones de selección .....       | 2         |
| 2.2.      | Creación de tablas con SELECT..... | 5         |
| 2.3.      | Inserción de datos con SELECT..... | 6         |
| <b>3.</b> | <b>Vistas.....</b>                 | <b>7</b>  |
| <b>4.</b> | <b>UPDATE.....</b>                 | <b>9</b>  |
| <b>5.</b> | <b>DELETE.....</b>                 | <b>10</b> |
| <b>6.</b> | <b>Transacciones.....</b>          | <b>11</b> |
| <b>7.</b> | <b>Resumen.....</b>                | <b>13</b> |

## 1. Introducción

En el tema anterior se estudiaron las instrucciones INSERT y SELECT. La instrucción SELECT es tan flexible y potente que además se puede utilizar:

- Para realizar inserciones de datos.
- Para crear tablas.
- Para crear vistas.

Para finalizar el estudio del sublenguaje DML, en este tema se tratan las instrucciones UPDATE y DELETE.

- UPDATE para actualizar los datos de una tabla.
- DELETE para borrar los datos de una tabla.

Una vez vistas todas las operaciones que incluye SQL para modificar el estado de la base de datos (insertar, modificar y borrar datos), se pueden abordar las transacciones de datos. Las transacciones son fundamentales para garantizar la integridad de datos cuando se realizan modificaciones, permitiendo confirmar o abortar los cambios realizados.

## 2. Otros usos de SELECT

Se verán a continuación otros usos de SELECT no vistos en el tema anterior.

### 2.1. Funciones de selección

Los SGBD incluyen muchas funciones predefinidas. La referencia completa de funciones que incluye MySQL se puede consultar en el siguiente enlace:

<https://dev.mysql.com/doc/refman/8.0/en/built-in-function-reference.html>

Unas funciones interesantes son las funciones de selección, que permiten establecer el resultado de una expresión en función de una o varias condiciones. Estas funciones son IF, IFNULL, NULLIF y CASE.

La función IF tiene la siguiente sintaxis:

```
IF(expresión, resultado1, resultado2)
```

donde:

- expresión: es una expresión booleana.
- resultado1: es el resultado de la función si expresión se evalúa a cierto y no es nulo.
- resultado2: es el resultado de la función si expresión es nula o falsa.

#### CASO PRÁCTICO 1

Si en la tabla CLIENTES se guarda un campo edad, para mostrar si un cliente es menor o mayor de edad:

```
SELECT nombre, apellidos, IF(edad < 18, 'Menor de edad', 'Mayor de edad')  
FROM clientes
```

La función IFNULL tiene la siguiente sintaxis:

```
IFNULL(expresión1, expresión2)
```

donde:

- expresión1: es el resultado de la función si se evalúa a cierto y no es nulo.
- expresión2: es el resultado de la función si expresión1 es nula o falsa.

## CASO PRÁCTICO 2

Para mostrar los teléfonos de los clientes, indicando si el cliente no ha informado teléfono:

```
SELECT nombre, apellidos, IFNULL(telefono, 'Sin teléfono')  
FROM clientes
```

En el resultado se muestran los teléfonos de los clientes que lo tengan informado. Para aquellos clientes sin teléfono, se mostrará el texto "Sin teléfono".

La función NULLIF tiene la siguiente sintaxis:

```
NULLIF(expresión1, expresión2)
```

La función devuelve:

- NULL si expresión1 es igual a expresión2.
- resultado1 si ambas expresiones son distintas.

## CASO PRÁCTICO 3

Si no se desea mostrar el teléfono 952 123 456 en una consulta:

```
SELECT nombre, apellidos, NULLIF(telefono, '952123456')  
FROM clientes
```

Para aquellos clientes que tengan asociado ese número de teléfono el resultado no lo mostrará y en su lugar aparece NULL.

La función CASE tiene dos formas de uso. La primera forma tiene la siguiente sintaxis:

```
CASE valor WHEN comparación THEN resultado  
  
[WHEN comparación THEN resultado ...]  
  
[ELSE resultado] END
```

La función compara el valor con cada uno de los valores de comparación, y devuelve el resultado asociado al primer valor donde la comparación es cierta. Si no hay ningún valor donde haya correspondencia, si se indica ELSE se devuelve el resultado asociado a ELSE.

#### **CASO PRÁCTICO 4**

Si en la tabla EMPLEADOS se guarda el código de departamento, y los departamentos son:

1 – Ventas

2 – Producción

3 – Recursos Humanos

4 – Dirección

Para mostrar nombre, apellidos y departamento:

```
SELECT nombre, apellidos, CASE departamento WHEN 1 THEN 'Ventas' WHEN 2 THEN  
'Producción' WHEN 3 THEN 'Recursos Humanos' WHEN 4 THEN 'Dirección' ELSE 'Otro'  
END  
FROM empleados
```

La segunda forma de CASE tiene la siguiente sintaxis:

```
CASE WHEN condición THEN resultado  
  
[WHEN condición THEN resultado ...]  
  
[ELSE resultado] END
```

La función devuelve el primer resultado donde condición se evalúe a cierto. Si no hay ninguna condición que sea cierta, si se indica ELSE se devuelve el resultado asociado a ELSE.

#### **CASO PRÁCTICO 5**

Se desea mostrar el rango de edad de los clientes:

```
SELECT nombre, apellidos, CASE WHEN edad<18 THEN 'Menor de edad' WHEN edad  
BETWEEN 18 and 25 THEN 'Entre 18 y 25' ELSE '26 años o superior' END  
FROM clientes
```

## 2.2. Creación de tablas con SELECT

Ya se vio que la sentencia DDL que permite crear tablas es CREATE TABLE. Es posible combinar CREATE TABLE y SELECT de tal modo que se cree una nueva tabla a partir de los resultados de una consulta. La sintaxis de creación de tabla es:

```
CREATE TABLE nombre [AS] SELECT ...
```

La tabla creada tiene el resultado de la consulta y los campos tienen los mismos tipos que los campos usados en la consulta. Los nombres también se mantienen, a menos que se indiquen alias para los campos en la consulta. Si en la consulta se utiliza alguna expresión, es muy recomendable indicar un alias para la expresión.

### IMPORTANTE

La tabla se crea sin clave primaria, ni ninguna restricción de clave foránea ni índice alguno. Si es necesario añadirlos deben realizarse luego modificaciones a la tabla.

### CASO PRÁCTICO 6

Imagina una tabla de oficinas y una de empleados. Se desea una tabla empleados\_oficinas\_esp para los empleados de España, donde aparece el nombre de la oficina de cada empleado y el nombre y apellidos del empleado deben aparecer concatenados en un único campo:

```
CREATE TABLE empleados_oficinas_esp AS
SELECT concat(e.nombre, e.apellido1, e.apellido2) as nombre_completo,
       o.nombre as oficina
FROM empleados e INNER JOIN oficinas o ON e.cod_oficina = o.cod_oficina
WHERE o.pais = 'España'
```

Este tipo de instrucciones son especialmente útiles para hacer copias de seguridad de datos. Por ejemplo, cuando es necesario modificar una tabla y resulta más cómodo borrarla (recuerda que DROP TABLE borra los datos que contenga la tabla) y crearla de nuevo.



## 2.3. Inserción de datos con SELECT

Ya se vio que la sentencia DML que permite insertar datos en una tabla es INSERT. Es posible combinar INSERT y SELECT para insertar datos en una tabla de la base datos. La sintaxis es:

```
INSERT [INTO] tabla[(campos...)] SELECT ...
```

Las columnas del resultado de la consulta deben concordar con la tabla en número y tipo.

### CASO PRÁCTICO 7

Imagina que hay una tabla llamada contador\_clientes donde se guardan el número de clientes de la empresa a final de cada mes. Cada final de mes es necesario guardar un nuevo contador de clientes en dicha tabla:

```
INSERT INTO contador_clientes(fecha, clientes)
SELECT curdate(), (SELECT count(*) FROM clientes)
```

### IMPORTANTE

Guardar en una tabla información que procede de otra puede parecer redundante y no tener sentido, especialmente cuando una de las ventajas de las bases de datos es la de evitar la redundancia.

Sin embargo, en casos como el ejemplo anterior es necesario guardar la información en otra tabla. Consultar el número de clientes de la tabla de clientes siempre da el valor actual, para consultar el número de clientes en meses pasados es necesario tenerlos almacenados en otra tabla.

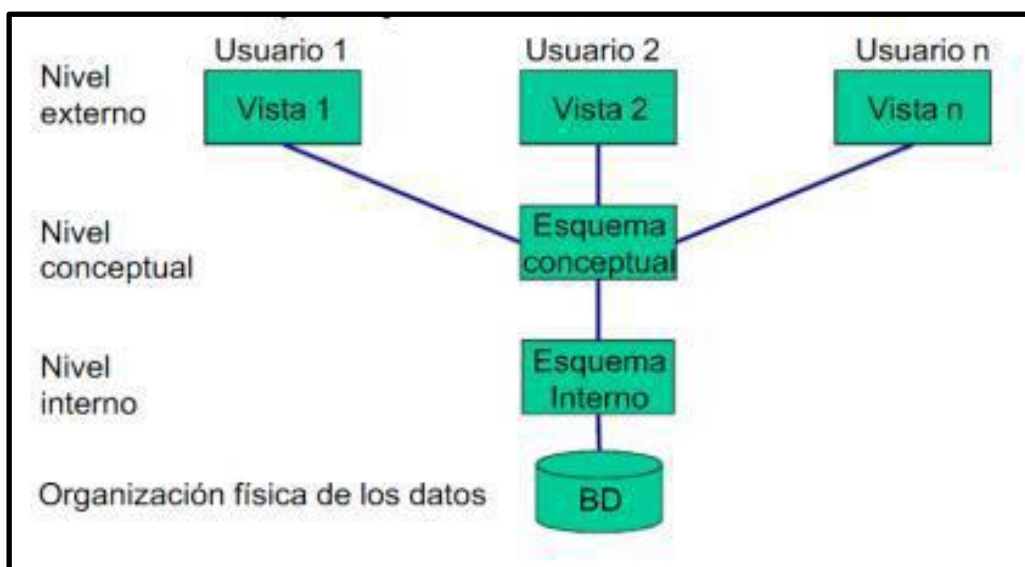
El uso de tablas de datos históricos también resulta útil para tablas que generan muchos datos. Puede ser necesario que, por razones de eficiencia, en la tabla solo se mantengan los valores de los últimos meses. Cuando un dato pasa ese umbral de tiempo se mueve a una tabla histórica para que pueda ser consultada cuando sea necesario, dejando en la tabla principal los valores más recientes.

### 3. Vistas

Una vista es una tabla virtual que devuelve el resultado de ejecutar una consulta SQL. La diferencia con una consulta ejecutada directamente es que, mientras cada consulta enviada al SGBD se valida y compila, la vista es una consulta cuya definición ha sido almacenada previamente y que está validada y compilada, siendo por tanto más rápida su ejecución.

Las vistas tienen una implicación importante para la seguridad. Un usuario podría no tener acceso a la información de varias tablas y, sin embargo, tener acceso a la vista que consulta esas tablas, proporcionando de esta manera un acceso controlado solo a determinadas filas y columnas de esas tablas. Las vistas son también útiles para que usuarios no expertos de la base de datos puedan acceder de forma fácil a la información, proporcionándoles a través de una vista la información obtenida a través de una sentencia SQL compleja.

En resumen, las vistas permiten configurar el nivel externo de la base de datos según la arquitectura ANSI/X3/SPARC, permitiendo así decidir la información de la base de datos que el usuario puede ver y de qué forma.



*Figura 1. Arquitectura ANSI/X3/SPARC.*

La creación de una vista sigue la siguiente sintaxis:

```
CREATE [OR REPLACE] VIEW nombre[(campos...)] AS SELECT ...
```

A continuación del nombre se puede indicar, de forma opcional, los campos que contendrán la vista. Si no se indican, se aprovechan los nombres de los campos que devuelve la consulta. Una vez creada la vista, ésta puede ser utilizada como cualquier otra tabla en una consulta.

Crear una vista con un nombre ya usado es un error, a menos que se indique la cláusula opcional `OR REPLACE`. En este caso, si la vista ya existe se reemplaza por la nueva definición.

Para modificar una vista se puede utilizar la cláusula `OR REPLACE`, o bien la instrucción:

```
ALTER VIEW nombre [(campos...)] AS SELECT ...
```

Para borrar una vista se utiliza la instrucción:

```
DROP VIEW nombre
```

## **CASO PRÁCTICO 8**

La tabla `empleados_oficinas_esp` del caso práctico 6 se podría haber creado como una vista del siguiente modo:

```
CREATE OR REPLACE VIEW empleados_oficinas_esp(nombre, oficina) AS  
SELECT concat(e.nombre, e.apellido1, e.apellido2), o.nombre  
FROM empleados e INNER JOIN oficinas o ON e.cod_oficina = o.cod_oficina  
WHERE o.pais = 'España'
```

Para obtener todos los empleados de las oficinas de España:

```
SELECT * FROM empleados_oficinas_esp
```

Si hiciera falta modificar la vista para añadir la región de la oficina, se puede realizar modificando la anterior instrucción y ejecutándola de nuevo, o con `ALTER VIEW`:

```
ALTER VIEW empleados_oficinas_esp(nombre, oficina, region) AS  
SELECT concat(e.nombre, e.apellido1, e.apellido2), o.nombre, o.region  
FROM empleados e INNER JOIN oficinas o ON e.cod_oficina = o.cod_oficina  
WHERE o.pais = 'España'
```

Finalmente, para eliminar la vista cuando ya no se necesite más:

```
DROP VIEW empleados_oficinas_esp
```

## 4. UPDATE

La sentencia UPDATE permite modificar el contenido de cualquier columna y de cualquier fila de una tabla. Su sintaxis es la siguiente:

```
UPDATE tabla  
SET columna1 = expresión1 [, columna2 = expresión2 ]  
[WHERE filtro]
```

La actualización se realiza dando a las columnas indicadas los valores de las expresiones. Se actualizan todas las filas de la tabla seleccionadas por el filtro indicado mediante la cláusula WHERE, siendo esta cláusula idéntica a la del SELECT. Se pueden incluir subconsultas en el filtro WHERE, con la limitación de que no puede incluirse la tabla que UPDATE pretende modificar.

### IMPORTANTE

Si se omite la cláusula WHERE todas las filas de la tabla se actualizan. Esto también aplica si el filtro indicado es cierto para todas las filas de la tabla.

Se debe evitar el uso de UPDATE sin indicar WHERE, ya que se corre el riesgo de *machacar* datos de la tabla que no se desean actualizar.

### IMPORTANTE

Recuerda que una clave primaria puede ser referenciada por una clave foránea en otra tabla y existir una restricción al actualizar la clave primaria (ON UPDATE RESTRICT o ON UPDATE NO ACTION). Por tanto, no siempre es posible actualizar el campo de una tabla.

### CASO PRÁCTICO 9

Se desea revisar el sueldo de los comerciales para incrementar un 15% el sueldo:

UPDATE empleados SET

sueldo = sueldo \* 1.15, fecha\_ultima\_revision = curdate()

WHERE departamento = 'Ventas'

## 5. DELETE

La sentencia DELETE permite eliminar filas de una tabla. Su sintaxis es:

```
DELETE FROM tabla  
[WHERE filtro]
```

La instrucción borra los registros seleccionados por el filtro WHERE, que es idéntico al de la sentencia SELECT. Se pueden incluir subconsultas en el filtro WHERE, con la limitación de que no puede incluirse la tabla donde DELETE pretende borrar.

### IMPORTANTE

Si se omite la cláusula WHERE se borran todas las filas de la tabla. Esto también aplica si el filtro indicado es cierto para todas las filas de la tabla.

**Es un error muy común omitir la cláusula WHERE al borrar datos. Este error puede tener consecuencias catastróficas si se pierden los datos.**

### IMPORTANTE

Al igual que en el caso de UPDATE, hay datos que están protegidos contra el borrado (ON DELETE RESTRICT o ON DELETE NO ACTION). Por tanto, no siempre es posible borrar una fila.

### DELETE vs TRUNCATE

Existen dos instrucciones para borrar todas las filas de una tabla:

```
DELETE FROM tabla
```

```
TRUNCATE TABLE tabla
```

Aunque puedan parecer iguales, realmente no lo son. DELETE se puede deshacer<sup>1</sup> mientras que TRUNCATE no.

### CASO PRÁCTICO 10

Se desean borrar los pedidos anteriores a 2010:

```
DELETE FROM pedidos
```

```
WHERE fecha_pedido < '2010-01-01'
```

---

<sup>1</sup> La forma en la que se deshace una inserción, modificación o borrado de datos se explica en el apartado de las transacciones.

## 6. Transacciones

Una transacción es un conjunto de instrucciones SQL que se tratan como una sola instrucción<sup>2</sup>. Una transacción puede acabar:

- Confirmada (COMMIT), si todas las operaciones individuales se ejecutaron correctamente.
- Abortada (ROLLBACK), si ocurre algún error que impide que todas las instrucciones se ejecuten correctamente. En este caso, los cambios realizados se deshacen.

Es decir, o se hacen todas las operaciones incluidas en la transacción, o no se hace ninguna. Las transacciones permiten mantener la integridad de los datos.

Por defecto, cuando se inicia sesión en MySQL está activado el modo autocommit, como se puede ver en los botones de opción que se aprecia en MySQL Workbench:



Cuando está activado el modo autocommit toda operación es automáticamente confirmada. Cada transacción está formada por una única instrucción.

Es posible desactivar el modo autocommit pulsando el citado botón de autocommit. En ese caso se activan los botones para confirmar (commit) y abortar (rollback):



Cuando el modo autocommit no está activado, los cambios que realice un usuario en la base de datos (inserción, actualización, borrado) serán vistos por el resto de usuarios una vez la transacción se confirme. Mientras tanto, el resto de usuarios verán el estado de la base de datos que había al comienzo de la transacción.

Una transacción comienza cuando ocurre uno de los siguientes eventos:

- El usuario inicia sesión en la base de datos.

---

<sup>2</sup> Se dice que el conjunto de instrucciones se ejecuta de forma atómica.

- Finaliza la transacción anterior, dando lugar a una nueva transacción.

Una transacción finaliza cuando ocurre uno de los siguientes eventos:

- El usuario finaliza la transacción, bien confirmando los cambios (COMMIT) o abortándolos (ROLLBACK), o realizando una operación DDL (CREATE, ALTER, DROP o bien TRUNCATE).
- El usuario cierra la sesión. Todos los cambios realizados son confirmados y en este caso no se da comienzo a una nueva transacción.

### **DELETE vs TRUNCATE**

En el apartado dedicado a DELETE se mostraron dos instrucciones para borrar los datos de una tabla, DELETE y TRUNCATE, y se comentó que TRUNCATE no se puede deshacer. Si dentro de una transacción se realiza un borrado, éste se puede deshacer realizando ROLLBACK. Sin embargo, TRUNCATE no se puede deshacer porque a todos los efectos es una operación DDL (TRUNCATE equivale a un DROP TABLE y CREATE TABLE para crear de nuevo la tabla).

La gestión de transacciones también se puede realizar con comandos SQL. Para crear una transacción se utiliza el comando BEGIN o START TRANSACTION, para confirmar el comando COMMIT y para abortar ROLLBACK.

### **CASO PRÁCTICO 11**

Para realizar un borrado que se pueda abortar es necesario iniciar una transacción:

START TRANSACTION

Para realizar el borrado:

DELETE FROM *tabla* WHERE *filtro*

Si hay que deshacer el borrado, se debe abortar la transacción:

ROLLBACK

Si hay que confirmar el borrado, se debe confirmar la transacción:

COMMIT

Recuerda que la confirmación también ocurre si se inicia una nueva transacción, si se realiza una instrucción DDL o si se cierra la sesión de base de datos.

## 7. Resumen

La instrucción `SELECT` es la más flexible y compleja de SQL. Además de ser utilizada en las consultas, se puede utilizar en combinación con `CREATE TABLE` para crear tablas y en combinación con `INSERT` para añadir registros a una tabla.

La instrucción `SELECT` también es usada para la creación de vistas, una herramienta poderosa para garantizar que los usuarios acceden únicamente a la información para la que tienen permisos o para facilitar la consulta de datos.

Además de gestionar las consultas a la base de datos, DML también se encarga de las inserciones, actualizaciones y borrados, gracias a las sentencias `INSERT`, `UPDATE` y `DELETE`, respectivamente.

Cuando se realizan cambios de datos en la base de datos, puede ser necesario realizar transacciones, es decir, secuencias de operaciones relacionadas que se tratan como una sola operación a efectos de la información resultante en la base de datos. Las transacciones son gestionadas por DCL, y se pueden confirmar o abortar. Las transacciones garantizan la fiabilidad e integridad de los datos almacenados en la base de datos.