

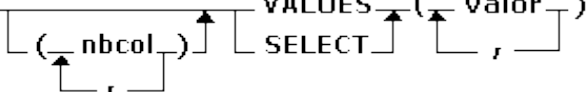
El lenguaje SQL-Actualización de datos.

Introducción.

- Hasta ahora hemos estudiado cómo recuperar datos almacenados en las tablas de nuestra base de datos. En este apartado vamos a tratar la actualización de esos datos, es decir, insertar nuevas tuplas o filas y borrar o cambiar el contenido de las filas de una tabla. Estas operaciones modifican los datos almacenados en las tablas, pero no la estructura de estas.

1 La sentencia INSERT INTO.

- La inserción de nuevos datos en una tabla se realiza añadiendo filas enteras a la misma. La sentencia SQL que lo permite es INSERT INTO. La inserción se puede realizar de una tupla o de varias filas a la vez.
- La sintaxis de esta sentencia es la siguiente:

INSERT INTO – destino 

- Esta sintaxis se utiliza para insertar una o varias filas cuyos valores indicamos después de la palabra reservada VALUES. También puede usarse en lugar de valores SET.
- *destino* es el nombre de la tabla donde vamos a insertar las filas, también se puede utilizar un nombre de consulta que tenga como origen de datos una única tabla. Al nombre de la tabla se le puede añadir la cláusula IN si ésta se encuentra en otra base de datos (en una base de datos externa).
- La palabra reservada VALUES se puede sustituir por la palabra SELECT (en muchos SQLs se permite únicamente VALUES).
- 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, entre # # o comillas simples para valores de fecha...) la asignación de valores se realiza por posición, el primer valor lo asigna a la primera columna, el segundo valor a la segunda columna, y así sucesivamente...
- Cuando la tabla tiene una columna de tipo contador (*Autoincrement*), lo normal es no asignar valor a esa columna para que el sistema le asigne el valor que le toque según el contador. Si queremos que la columna tenga un valor concreto lo indicamos en la lista de valores y el próximo insert que hagamos sin especificar un valor comenzará por el valor anterior + 1.

- 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 tabla, y se tiene que utilizar el valor NULL para rellenar las columnas de las cuales no tenemos valores.

Ejemplo 1: A continuación podemos observar la forma de insertar un nuevo registro en la tabla empleado. En este caso no se especifican los nombres de los campos, por lo que se debe indicar un valor para cada uno de ellos, aunque éste sea NULL.

```
INSERT INTO empleado VALUES (200, 'Juan López', #06/23/01#, 30, 'rep  
ventas', NULL, NULL, 350000, 0);
```

- Cuando indicamos los nombres de las columnas, estos no tienen por qué estar en el orden en que aparecen en la tabla, pudiéndose, incluso, omitir algunas columnas, en cuyo caso tendrán por defecto el valor NULL o el valor predeterminado del campo.

Ejemplo 2: El ejemplo anterior se podría escribir de la siguiente forma:

```
INSERT INTO empleado (numemp, codoficina, nombre, cargo, cuota, fcontrato,  
ventas) VALUES (200, 30, 'Juan López', 'rep ventas', 350000, #06/23/01#, 0);
```

- El uso de la opción de poner una lista de columnas podría parecer peor, ya que se tiene que escribir más, pero realmente tiene las siguientes ventajas:
 - ▲ La sentencia resulta más fácil de interpretar.
 - ▲ Nos aseguramos que el valor lo asignamos a la columna que queremos.
 - ▲ Si por alguna razón se cambia el orden de las columnas o se añade o borra una columna la sentencia seguiría funcionando.
- Errores que se pueden producir al ejecutar una 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.
 - ▲ Si tenemos definido un índice único (sin duplicados) e intentamos asignar un valor que ya existe en la tabla.

IMPORTANTE: *En MySql las columnas UNIQUE admiten NULL si no están definidas como NOT NULL.*

- ▲ Si la tabla está relacionada con otra se seguirán las *reglas de integridad referencial*.
- La sentencia INSERT INTO permite añadir, en una sola sentencia, varias filas a la vez, para lo cual encerraremos el conjunto de registros entre paréntesis.

Ejemplo 3: Inserción de dos registros usando una única sentencia INSERT INTO.

```
INSERT INTO empleado (numemp, codoficina, nombre, cargo, cuota, fcontrato, ventas)  
VALUES (500, 33, 'Antonio Romero', 'rep ventas', 350500, '05/09/06', 0), (550, 40,  
'María Ruiz', 'rep ventas', 450000, '02/11/05', 0);
```

Ejercicio 1: (INSERT INTO) Realiza las siguientes inserciones de filas en la relación *empleado* de la BD Ventas:

1. Los datos de un empleado cuyo código será el 111, su nombre es *Amadeo Benítez López*, su cargo es representante, su fecha de nacimiento '12/09/1965', su jefe será el empleado 104 y su contrato tiene la fecha de hoy. Aún no se ha asignado a ninguna oficina ni se han establecido cuota ni ventas.
2. Los datos de una nueva oficina en la ciudad de *Málaga*, región sur. El director será el empleado 105. Su código será 29. Aún no se conocen más datos de la misma.
3. Los datos de un nuevo producto, añadiendo, en este mismo orden, el precio será 200, la descripción *manivela*, su código 123 y sus existencias 20.
4. Inserta en las tablas de pedidos y líneas de pedido, un pedido número 11223, fecha actual, realizado por el cliente 2106. El producto pedido es el de código 773c. Se piden 10 unidades del mismo, con un importe de 9750.

2 Integridad referencial.

- En este apartado recordaremos la *regla de integridad referencial*, y analizaremos cómo debemos tenerla en cuenta a la hora añadir, modificar o eliminar datos de las tablas.
- Como sabemos, la integridad referencial es un sistema de reglas que utilizan las bases de datos relacionales para asegurar en todo momento que los registros de tablas relacionadas son válidos, evitando que se borren o cambien datos relacionados de forma accidental produciendo errores de integridad.
- Recordemos que cuando se define una columna como clave foránea, las filas de la tabla pueden contener en esa columna o bien el valor nulo, o bien un valor que existe en la otra tabla. La integridad referencial hace que el sistema gestor de la base de datos se asegure de que no hayan en las claves foráneas valores que no estén en la tabla principal.
- La integridad referencial se activa cuando creamos una clave foránea, y a partir de ese momento se comprueba cada vez que se modifiquen datos que puedan alterarla.
- La violación de la regla de integridad referencial puede producirse en los siguiente casos:

- Cuando insertamos una nueva fila en la tabla secundaria y el valor de la clave foránea no existe en la tabla principal.
 - Cuando modificamos el valor de la clave principal de un registro que tiene filas relacionadas en otra tabla. En este caso hay que decidir qué hacer con estas filas relacionadas.
 - Cuando modificamos el valor de la clave foránea, ya que el nuevo valor debe existir en la tabla principal.
 - Cuando queremos borrar una fila de la tabla principal y ese registro tiene filas relacionadas en otra tabla.
- Asociados a la regla de integridad referencial tenemos los conceptos de *actualizar los registros en cascada* y *eliminar registros en cascada*.
 - **Actualizar y/o eliminar registros en cascada (Update Cascade, Delete Cascade)**, son opciones que se deben definir cuando definimos la clave foránea y que le indican al sistema gestor qué hacer en los casos comentados en el punto anterior.
 - La opción *actualizar registros en cascada* indica al sistema gestor de la base de datos que cuando se cambie un valor del campo clave de la tabla principal, automáticamente se cambiará el valor de la clave foránea de los registros relacionados en la tabla secundaria.

Opciones Clave Foránea:

- NO ACTION no se puede modificar o borrar el padre si tiene hijos.
- RESTRICT = NO ACTION
- CASCADE si se borra o actualiza el padre se opera igual en los hijos.
- SET NULL Si se actualiza o borra el padre, se ponen a NULL las claves ajenas en los hijos que deben estar definidas con la opción NULL.

Ejemplo 4: Si cambiamos en una tabla de *poblaciones* (la tabla principal) el valor 1 por el valor 10 en el campo *codigo* (la clave principal), automáticamente se actualizan todos los habitantes (en la tabla secundaria) que tienen el valor 1 en el campo *poblacion* (en la clave ajena) dejando 10 en vez de 1. Si no se tiene definida esta opción, no se puede cambiar los valores de la clave principal de la tabla principal. En este caso, si intentamos cambiar el valor 1 del *codigo* de la tabla de *poblaciones*, no se produce el cambio y el sistema nos devuelve un error.

- La opción *eliminar registros en cascada* indica al sistema gestor de la base de datos que cuando se elimina un registro de la tabla principal, automáticamente se borran también los registros relacionados en la tabla secundaria.

Ejemplo 5: Si borramos la población *Cádiz* en la tabla de poblaciones, automáticamente todos los habitantes de *Cádiz* se borrarán de la tabla de habitantes. Si no se tiene definida esta opción, no se pueden borrar registros de la tabla principal si estos tienen registros relacionados en la tabla secundaria. En este caso, si intentamos borrar la población *Cádiz*, no se produce el borrado y el sistema nos devuelve un error.

3 INSERT INTO ... SELECT.

- Podemos insertar filas en una tabla con una sentencia SELECT INTO. En este caso, los valores a insertar se deben obtener como resultado de una consulta, y sustituimos la cláusula VALUES *lista de valores* por una sentencia SELECT común. Cada fila resultado de la sentencia SELECT forma una lista de valores que son los que se insertan en una nueva tupla de la tabla destino.
- La sintaxis de esta sentencia es la siguiente:

INSERT INTO – destino SELECT ...
(nbcolumna)
,

- El origen de la sentencia SELECT puede ser el nombre de una consulta guardada, un nombre de tabla o una composición de varias tablas mediante INNER JOIN, LEFT JOIN, RIGHT JOIN o producto cartesiano.
- Cada fila devuelta por la sentencia SELECT actúa como la lista de valores que vimos con la INSERT INTO, por lo que tiene las mismas restricciones en cuanto a tipo de datos, etc. La asignación de valores se realiza por posición, por lo que la sentencia 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 sentencia SELECT no tienen por qué llamarse igual que en la tabla destino, ya que el sistema sólo se fija en los valores devueltos.
- Si no queremos asignar valores a todas las columnas entonces tenemos que indicar entre paréntesis la lista de columnas a rellenar después del nombre del destino.
- El estándar ANSI/ISO para SQL especifica varias restricciones sobre la consulta que aparece dentro de la sentencia INSERT. En concreto:
 - ▲ La consulta no puede tener una cláusula ORDER BY.
 - ▲ 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.

Ejemplo 6: Supongamos que tenemos una tabla llamada *repres* con la misma estructura que la tabla *empleado*, y queremos insertar en esa tabla los empleado que tengan como cargo *rep ventas*.

```
INSERT INTO repres SELECT * FROM empleado
WHERE cargo = 'representantes';
```

Ejemplo 7: Supongamos ahora que la tabla *repres* tuviese las siguientes columnas: *numemp*, *oficinarep*, *nombrerep*. En este caso no podríamos utilizar el asterisco, tendríamos que poner:

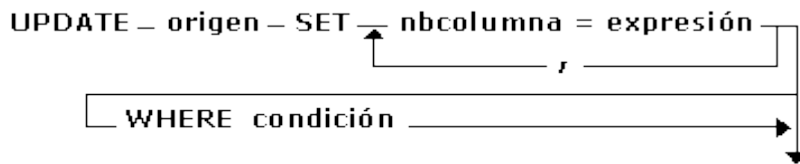
```
INSERT INTO repres SELECT numemp, codoficina, nombre FROM empleado
WHERE cargo = 'rep ventas';
```

Ejercicio 2: (INSERT INTO SELECT) Realiza las siguiente operaciones:

- Los datos de un empleado cuyo código será el 111, su nombre es *Benítez López*, Amadeo su cargo es representante, su fecha de nacimiento '12/09/1965', su jefe será el empleado López Gómez, José y su contrato tiene la fecha de hoy. Aún no se ha asignado a ninguna oficina ni se han establecido cuota ni ventas.
- Crea una nueva tabla llamada *InformeVentas* en la BD *Ventas*. Dicha tabla estará compuesta por un campo de tipo texto llamado *Empleado*, otro de tipo numérico llamado *CodOficina*, y dos campos de tipo entero llamados *VentasEmpleados* y *VentasOficinas*. No es necesario definir ninguna clave principal para esta tabla.
- Inserta en la nueva tabla todos los empleados existentes en la tabla empleado con sus códigos de oficinas correspondientes y con sus *VentasOficina* y *Ventasempleado* con un valor de 50000 euros cada una de ellas.
- Insertar en la tabla *InformeVentas* los empleados de la oficina 50, sus datos serán el nuevo código de oficina 80 y sus ventas de empleado correspondientes.
- Inserta en la tabla anterior los nombres de los empleados, los códigos de sus oficinas y las ventas del mismo y de su oficina, para aquellos empleados que hayan superado su cuota de ventas y tengan jefe.
- Añade a la tabla anterior los nombres de los empleados que sean representantes de algún cliente y las ventas de sus oficinas.
- Añade a la tabla anterior los nombres de los empleados, los códigos de sus oficinas y las ventas de su oficina, para aquellos empleados cuyos clientes hayan pedido alguna vez un *reostato* o una *red*.
- Crea una nueva tabla llamada *ResumenEmpleados* en la BD *Ventas*. Dicha tabla estará compuesta por un campo de tipo texto llamado *Empleado* y otro de tipo numérico llamado *CodOficina*. No es necesario definir ninguna clave principal para esta tabla.
- Inserta en esta tabla los nombres de empleados y códigos de oficina de la tabla *InformeVentas*, de todos los empleados cuyas ventas supongan más de la mitad de las ventas de su oficina y su contrato sea anterior a 1989.
- Añade a esta tabla los nombres de **todos** los representantes cuyos clientes hayan realizado algún pedido con un importe mayor de 5000.

4 Modificación del contenido de las tablas. Sentencia UPDATE.

- La sentencia UPDATE modifica los valores de una o más columnas en las filas seleccionadas de una o varias tablas.
- La sintaxis de esta sentencia es la siguiente:



- *origen* puede ser un nombre de tabla, un nombre de consulta o una composición de tablas, también puede incluir la cláusula IN si la tabla a modificar se encuentra en una base de datos externa.
- La cláusula SET especifica qué columnas van a modificarse y qué valores se asignarán a las mismas.
- *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.
- *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. ***expresión* puede ser una subconsulta pero NO puede estar referida a la tabla sobre la que se realiza el UPDATE.**

Ejemplo 8: Queremos actualizar las cuotas de nuestros empleados elevándola un 20%.

```
UPDATE empleado set cuota=cuota*1.20;
```

- 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 estándar **la tabla que aparece tras la sentencia FROM de la subconsulta no puede ser la misma que la tabla que aparece como origen del UPDATE.**

Ejemplo 9: Queremos poner a cero las ventas de los empleados de la oficina 12.

```
UPDATE empleado SET ventas = 0 WHERE codoficina = 12;
```

Ejemplo 10: Queremos poner a cero el límite de crédito de los clientes asignados a empleados de la oficina 12.

```
UPDATE cliente SET limcredito = 0
WHERE repclie IN (SELECT numemp FROM empleado
WHERE codoficina = 12);
```

Ejemplo 11: Modificar el límite de crédito del cliente con número de cliente 30 para que tenga el mismo límite de crédito del cliente 20.

```
UPDATE cliente SET limcredito = (Select limcredito FROM cliente
WHERE numclie= 20) WHERE numclie=30;
```

En Oracle esta orden no daría ningún problema pero en MySQL sí, porque el SQL estándar no se permite que la tabla del Update se use dentro de las Subselect.

- Si para el cálculo de *expresión* se utiliza una columna que también se modifica, el valor que se utiliza es el de antes de la modificación, igual que 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 a la vez todas las columnas incluidas en la cláusula SET, por lo que los nombres de columna pueden especificarse en cualquier orden; después se coge la siguiente fila de la selección y se procede del mismo modo, y así sucesivamente con todas las filas de la selección.

Ejemplo 12: Modificación de varias columnas a la vez. Ambas sentencias son equivalentes.

UPDATE oficina SET ventas=0, objetivo=ventas; -- ventas=0, objetivo=0

O bien:

UPDATE oficina SET objetivo=ventas, ventas=0; -- ventas =objetivo, ventas=0

- Si actualizamos una columna definida como clave foránea, esta columna se podrá actualizar, o no, siguiendo las reglas de integridad referencial. El valor que se le asigna debe existir en la tabla de referencia.
- Si actualizamos una columna definida como columna principal de una relación entre dos tablas, esta columna se podrá actualizar, o no, siguiendo las reglas de integridad referencial.

Ejercicio 3: (UPDATE) Abre la BD *Ventas* y realiza las siguientes operaciones:

- a) Añadir una nueva tabla, llamada *ejercicio31*, a la BD *Ventas*. Esta nueva tabla contendrá los códigos de todos los productos existentes, sus precios y sus existencias, así como el número total de unidades vendidas. A este último campo se llamará *cantidadvendida* con valor 0 por defecto si el producto no se ha vendido nunca.
- b) Cambia todos los valores del campo *cantidadvendida* de la tabla anterior que sean menores que 10 por el valor 0.
- c) Incrementa el nivel de existencias de todos los productos cuyo código comience por 41 en 25 unidades.
- d) Rebaja el precio a la mitad de los artículos cuyo código comienza por 2. No se tocará el precio de un artículo si ya es menor de 100 €.
- e) Incrementa en un 15% el precio de todos los productos de la tabla *ejercicio31* para los que la cantidad vendida suponga al menos el 20% de las existencias actuales.
- f) Rebaja en un 5% el precio de todos los productos que cuesten menos de 1000 € y cuyo nivel de existencias sea mayor del doble de la media de las existencias de todos los productos.
- g) Resta 10 al campo *cantidadvendida* de todos los productos que tengan un valor en este campo mayor que 10 y para los cuales se hayan realizado al menos dos ventas.

6 Eliminación de filas. Sentencia DELETE.

- La sentencia DELETE elimina filas de una tabla. Su sintaxis es la siguiente:

DELETE ----- FROM origen -----
-- tabla.*-- --- WHERE condición----

- origen* es el nombre de la tabla de la que vamos a eliminar las filas. Es posible indicar un nombre de tabla, incluir la cláusula IN si la tabla se encuentra en una base de datos externa y también podemos escribir una composición de tablas.
- La opción *tabla.** se utiliza cuando el origen está basado en varias tablas, y sirve para indicar de qué tabla vamos a borrar.
- La cláusula WHERE sirve para especificar qué filas queremos borrar. Se eliminarán de la tabla todas las filas que cumplan la condición. Si no se indica la cláusula WHERE entonces se borran todas las filas de la tabla.
- En la condición de búsqueda de la sentencia DELETE se puede utilizar una subconsulta. Las tablas que aparecen en la cláusula FROM de la subconsulta no pueden ser la misma que la tabla que aparece en la cláusula FROM de la sentencia DELETE.

Ejemplo 13: Las dos sentencias siguientes eliminan los pedidos del cliente *López, Julián*. En la segunda estamos obligados a escribir *pedido.**, ya que el origen está basado en varias tablas.

DELETE FROM pedido WHERE cliente = (SELECT numclie FROM cliente WHERE nombre = 'López, Julián');

DELETE pedido. FROM pedido, cliente where cliente = numclie and cliente.nombre = 'López, Julián';*

Usando alias de tablas sería:

DELETE p. FROM pedido p, cliente c where cliente = numclie and c.nombre = 'López, Julián';*

Ejemplo 14: La siguiente sentencia elimina todas las filas de la tabla *pedido*.

DELETE FROM pedido;

Ejercicio 4: (DELETE) Realiza las siguientes operaciones:

- Elimina todos los registros de la tabla *ejercicio31* para los que la cantidad vendida sea menor de 8.
- Elimina todos los productos de la tabla *ejercicio31* que alguna vez fueron pedidos por clientes con un representante de la zona oeste.
- Elimina los productos de la tabla *ejercicio31* que nunca fueron vendidos.
- Elimina todos los productos de la tabla *ejercicio31* que sean *rodamientos*, *reostatos* o *redes*.

- e) Elimina todos los registros que queden en la tabla *ejercicio31*.
- f) Elimina la tabla *ejercicio31*.

Ejercicio 5: (Integridad referencial) Crea una BD llamada *DatosAlumnos* compuesta por las dos siguientes tablas:

Padres:

NombPadre	ApellPadre	DNI (*)
Juan	Romero Reyes	25707887
Antonio	Merino Jiménez	20090887
Pedro Luis	Torres Olmo	32009007

Alumnos:

NombAlum	ApellAlumno	DNIALum (*)	DNIPadre
Luisa	Torres Moreno	66789987	32009007
María	Merino Gómez	76678777	20090887
Rocío	Romero González	45554333	25707887
Marta	Merino Gómez	58767890	20090887
Ismael	Torres Moreno	54432213	32009007
José Pedro	Torres Moreno	90008768	32009007

- a) Añade los datos de un alumno llamado *Rafael Gutiérrez Moreno* con DNI 33456540. El DNI de su padre será 41443332. Analiza el resultado obtenido al intentar realizar esta inserción.
- b) Intenta eliminar el registro que contiene los datos del padre llamado *Antonio Merino Jiménez*. Analiza el resultado obtenido.
- c) Intenta cambiar el valor del campo DNI del padre *Juan Romero Reyes* por 77665544. Analiza el resultado obtenido.
- d) Modifica la clave foránea para actualizar en cascada los campos relacionados y eliminar en cascada los registros relacionados.
- e) Repite nuevamente el apartado b. ¿Qué ha sucedido?
- f) Repite nuevamente el apartado c) ¿Qué ha sucedido?
- g) Elimina lo que se hizo en el apartado d), es decir que no se actualice en cascada ni que tampoco se elimine en cascada.