UNIDAD DIDÁCTICA 7. DML (LENGUAJE DE MANIPULACIÓN DE DATOS)

UNIDAD DIDÁCTICA 7. DML (LENGUAJE DE MANIPULACIÓN DE DATOS)

Si no funcionan las sentencias UPDATE, DELETE hay que actualizar el valor de la siguiente variable:

```
SET SQL_SAFE_UPDATES = 0
```

El lenguaje DML permite la **actualización** de los 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**). Si trabajamos en un **entorno multiusuario**, todas estas operaciones se podrán realizar **siempre que tengamos los permisos correspondientes**.

INSERT INTO…VALUES Inserción de nuevas filas

La forma más directa de insertar una fila nueva en una tabla es mediante una sentencia **INSERT**. En la forma más simple de esta sentencia debemos indicar la tabla a la que queremos añadir filas, y los valores de cada columna. Las columnas de tipo cadena o fechas deben estar entre comillas sencillas o dobles, para las columnas numéricas esto no es imprescindible, aunque también pueden estar entrecomilladas.

La **sintaxis** es la siguiente:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
VALUES ({expression | DEFAULT},...),(...),...
[ ON DUPLICATE KEY UPDATE col_name=expression, ... ]
```

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 ….**

Los registros **se agregan** siempre **al final de la tabla**.

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 # # 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, así sucesivamente...

Cuando la tabla tiene una **columna de tipo contador** (Auto_increment), 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 rellenar las columnas de las cuales no tenemos valores.

Ejemplo: Insertar un futbolista en la tabla Futbolistas

```
INSERT INTO Futbolistas (numFicha, nombre, apellidos, fecNacimiento,
peso, altura) VALUES
(200, 'Juan', 'López Pereira', '1992/01/26', 82, 185);
INSERT INTO Futbolistas (numFicha, nombre, apellidos, fecNacimiento,
peso, altura) VALUES
(201, 'Pedro', 'Ormaechea García', '1990/05/06', 85, 180),
(202, 'Jorge', 'Pastoriza González', '1988/10/16', 92, 195);
```

Observar en el ejemplo que los valores de tipo texto y fecha se encierran entre comillas simples ' ' (también se pueden emplear las comillas dobles " ") con el formato año/mes/dia.

Cuando **indicamos nombres de columnas**, estos corresponden a nombres de **columna de la tabla**, pero no tienen por qué estar en el **orden** en que aparecen en la ventana diseño de la tabla.

Ejemplo: Insertar un futbolista en la tabla Futbolistas

```
INSERT INTO Futbolistas (numFicha, fecNacimiento, peso, altura,
nombre, apellidos) VALUES
(221, '1991/07/25', 75, 183, 'Miguel', 'Sacristán Duro'),
(222, '1988/10/22', 87, 190, 'Iñigo', 'Berraondo Aguirre');
```

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.

También **se pueden omitir algunas columnas**, la columnas que no se nombran tendrán **por defecto el valor NULL o el valor predeterminado** indicado en la ventana de diseño de tabla.

Ejemplo: Cambiar el tipo de dato idIdioma a VARCHAR(15) valor por Defecto Español en la tabla libros. Insertar un libro en la tabla Libros

```
ALTER TABLE Libros MODIFY idIdioma VARCHAR(15) DEFAULT 'Español';
INSERT INTO Libros (idLibro, titulo, anhoPublicacion, diasMaxPrestamo)
VALUES
(25, 'El Quijote', 2019, 15);
```

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**:

- la sentencia queda **más fácil de interpretar** leyéndola vemos qué valor asignamos a qué columna,
- de paso nos **aseguramos** que el valor lo asignamos a la columna que queremos,
- **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,

- otra ventaja es que **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.

Si no se pone la lista con los nombres de las columnas hay que insertar todos los valores, incluidos los nulos

Ejemplo: Insertar un libro en la tabla Libros

```
INSERT INTO Libros VALUES
(30, 'Bases de Datos','Inglés', null, null, null, 2000, 20),
(31, 'MySQL', 'Inglés', null, null, null, 2003, 22),
(32, 'Código Limpio', null, null, null, null, 2001, 18);
```

Ejemplo: Insertar valores autonuméricos.

```
#Crear la tabla Discos
CREATE TABLE Discos(
codDisco INT AUTO_INCREMENT PRIMARY KEY,
titulo VARCHAR(20),
precio INT DEFAULT 25);
```

1º) Insertamos un disco sin poner los nombres de las columnas

```
INSERT INTO Discos VALUES
(null, 'Please Please Me' , 30),
(null, 'Thriller', 25);
INSERT INTO Discos VALUES
(100, 'Back in Black', 20),
(null, 'The Dark Side Of The Moon', 24);
```

2º) Insertamos un disco poniendo los nombres de las columnas, omitimos el nombre de la columna autonumérica.

```
INSERT INTO Discos (titulo, precio) VALUES
('The Bodyguard', 19),
('El fantasma de la ópera', 23);
```

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 de '*infracciones de clave*'.
- **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**.

Insertar varias filas INSERT INTO…SELECT

Con **INSERT … SELECT**, se pueden insertar rápidamente muchas filas en una tabla desde otra u otras tablas.

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** como las que hemos visto hasta ahora. **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 [LOW_PRIORITY] [IGNORE] [INTO] tbl_name [(column_list)]
SELECT ...
```

El **origen** de la **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 **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 por qué 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 rellenar **después del nombre del destino.**

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.

Ejemplo: Supongamos que tenemos una tabla llamada Copias con la misma estructura que la tabla Discos, y queremos insertar en esa tabla los discos que tengan un precio mayor que 20.

```
#Crear la tabla Copias
CREATE TABLE Copias(
codDisco INT,
titulo VARCHAR(30),
```

```
precio INT DEFAULT 25);
INSERT INTO Copias SELECT * FROM Discos WHERE precio > 20;
```

Con la **SELECT** obtenemos las filas correspondientes a los discos con precio > 20 y las insertamos en la tabla *Copias*. Como las tablas tienen la misma estructura no hace falta poner la lista de columnas y podemos emplear * en la lista de selección de la **SELECT**.

Ejemplo: Supongamos ahora que la tabla Copias tuviese las siguientes columnas codCod, copTitulo.

En este caso las columnas no tienen el mismo nombre y queremos copiar todas por lo que no podríamos utilizar el asterisco, tendríamos que poner:

```
#Crear la tabla Copias1
CREATE TABLE Copias1(
copCod INT,
copTitulo VARCHAR(30));
INSERT INTO Copias1 (copCod, copTitulo)
SELECT codDisco, titulo FROM Discos WHERE precio > 23;
```

O bien:

```
INSERT INTO Copias1 SELECT codDisco, titulo FROM Discos
WHERE precio > 23;
```

Insertar filas en una nueva tabla

Esta sentencia **inserta filas creando** en ese momento **la tabla donde se insertan** las filas. Se suele utilizar **para guardar en una tabla el resultado de una SELECT**.

La **sintaxis** es la siguiente:

```
CREATE TABLE nbtabla……SELECT….
```

- **Las columnas de la nueva tabla** tendrán el **mismo tipo y tamaño que las columnas origen**, y **se llamarán con el nombre de alias** de la columna origen o **en su defecto con el nombre de la columna origen**, pero **no se transfiere ninguna otra propiedad** del campo o de la tabla **como por ejemplo las claves e índices.**
- La sentencia **SELECT** puede ser **cualquier sentencia SELECT sin ninguna restricción**, puede ser una consulta multitabla, una consulta de resumen, una **UNION**

Ejemplo:

```
CREATE TABLE CopiaLibros SELECT * FROM Libros;
```

Esta sentencia genera una nueva tabla **CopiaLibros** con todas las filas de la tabla **Libros**. Las columnas se llamarán igual que en **Libros** pero **CopiaLibros** no será una copia exacta, ya no tendrá clave principal ni relaciones con las otras tablas, ni índices si los tuviese **Libros** etc...

- **Si en la base de datos hay ya una tabla del mismo nombre**, el sistema **nos da un mensaje de error** y no se ejecuta.
- **Para** formar **una sentencia CREATE TABLE…… SELECT** lo mejor es **escribir la SELECT** que permite generar los datos que queremos guardar en la nueva tabla, y **después añadir** delante de la cláusula **FROM** la cláusula **INTO nuevatabla**.

 La sentencia **CREATE TABLE…… SELECT se suele utilizar para crear tablas de trabajo**, o tablas intermedias, las creamos para una determinada tarea y cuando hemos terminado esa tarea las borramos. También puede ser útil **para sacar datos en una tabla para enviarlos a alguien**.

UPDATE Modificar el contenido de las filas

SET SQL_SAFE_UPDATES=0;

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

La **sintaxis** es la siguiente:

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
SET col_name1=expr1 [, col_name2=expr2 ...]
[WHERE where_definition]
[ORDER BY ...]
[LIMIT row_count]
```

tbl_name 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**.

col_name, 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.

expr1 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**. **expr1** no puede ser una subconsulta.

Ejemplo: Queremos incrementar el precio de los discos en 5 euros

```
UPDATE Discos SET precio = precio + 5;
```

Ejemplo: Queremos incrementar el precio de los discos que tienen un código menor que 100 en 10 euros

```
UPDATE Discos SET precio = precio + 10 WHERE codDisco <100;
```

Ejemplo: Queremos incrementar el precio de los discos en 50 euros y poner el título aaaaaa

```
UPDATE Discos SET precio = precio +50, titulo = 'aaaaaa';
```

- 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 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.**

DELETE Borrar filas

La sentencia **DELETE elimina filas de una tabla**.

La **sintaxis** es la siguiente:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM table_name
[WHERE where_definition]
[ORDER BY ...]
[LIMIT row_count]
```

La forma más simple es no usar ninguna de las cláusulas opcionales:

Ejemplo: Eliminar todas las filas de la tabla Discos

```
DELETE FROM Discos;
```

Pero es más frecuente que sólo queramos eliminar ciertas filas que cumplan determinadas condiciones. La forma más normal de hacer esto es usar la cláusula ***WHERE***:

Ejemplo: Eliminar las filas de la tabla Copias cuyo precio sea 25 o mayor

```
DELETE FROM Copias WHERE codDisco >=25;
```

También podemos usar las cláusulas *LIMIT* y *ORDER BY* del mismo modo que en la sentencia **UPDATE.**

Ejemplo: Eliminar los 2 libros más antiguos de la tabla CopiaLibros

```
DELETE FROM CopiaLibros ORDER BY AnhoPublicacion LIMIT 2;
```

Si se usa una sentencia **DELETE** sin cláusula ***WHERE***, todas las filas serán borradas. Una forma más rápida de hacer esto, cuando no se necesita conocer el número de filas eliminadas, es usar **TRUNCATE TABLE**.

Si se borra la fila que contiene el valor máximo para una columna *AUTO_INCREMENT*, ese valor podrá ser usado por una tabla **ISAM** o **BDB**, pero no por una tabla **MyISAM** o **InnoDB**. Si se borran todas las filas de una tabla con **DELETE FROM tbl_name** (sin un WHERE) en modo *AUTOCOMMIT*, la secuencia comenzará de nuevo para todos los motores de almacenamiento, excepto para **InnoDB**.

La sentencia **DELETE** soporta los siguientes modificadores:

- Si se especifica la palabra *LOW_PRIORITY*, la ejecución de **DELETE** se retrasa hasta que no existan clientes leyendo la tabla.
- Para tablas **MyISAM**, si se especifica la palabra *QUICK* entonces el motor de almacenamiento no mezcla los permisos de índices durante el borrado, esto puede mejorar la velocidad en ciertos tipos de borrado.
- La opción *IGNORE* hace que MySQL ignore todos los errores durante el proceso de borrado. (Los errores encontrados durante en análisis de la sentencia se procesan normalmente.) Los errores que son ignorados por el uso de esta opción se devuelven como avisos.

- La velocidad de las operaciones de borrado pueden verse afectadas por otros factores, como el número de índices o el tamaño del caché para índices.
- En tablas **MyISAM**, los registros borrados se mantienen en una lista enlazada y subsiguientes operaciones **INSERT** hacen uso de posiciones anteriores. Para recuperar el espacio sin usar y reducir el tamaño de los ficheros, usar la sentencia **OPTIMIZE TABLE** o la utilidad **myisamchk** para reorganizar tablas. **OPTIMIZE TABLE** es más fácil, pero **myisamchk** es más rápido.
- El modificador *QUICK* afecta a si las páginas de índice se funden para operaciones de borrado. **DELETE QUICK** es más práctico para aplicaciones donde los valores de índice para las filas borradas serán reemplazadas por valores de índice similares para filas insertadas más adelante. En ese caso, los huecos dejados por los valores borrados serán reutilizados.
- **DELETE QUICK** no es práctico cuando los valores borrados conducen a bloques de índices que no se llenan dejando huecos en valores de índice para nuevas inserciones. En ese caso, usar *QUICK* puede dejar espacios desperdiciados en el índice que permanecerán sin reclamar. Para liberar el espacio de índices no usado bajo estas circunstancias, se puede usar **OPTIMIZE TABLE**.
- Se pueden especificar múltiples tablas en la sentencia **DELETE** para eliminar filas de una o más tablas dependiendo de una condición particular en múltiples tablas. Sin embargo, no es posible usar *ORDER BY* o *LIMIT* en un **DELETE** multitabla.
- Una vez borrados, **los registros no se pueden recuperar**.
- **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

TRUNCATE Vaciar una tabla

Cuando queremos eliminar todas las filas de una tabla, vimos en el punto anterior que podíamos usar una sentencia **DELETE** sin condiciones. Sin embargo, existe una sentencia alternativa, **TRUNCATE**, que realiza la misma tarea de una forma mucho más rápida.

```
TRUNCATE TABLE table_name
```

La diferencia es que **DELETE** hace un borrado secuencial de la tabla, fila a fila. Pero **TRUNCATE** borra la tabla y la vuelve a crear vacía, lo que es mucho más eficiente.

Ejemplo: eliminar la tabla Contrato

```
TRUNCATE contrato;
```