T09. DML: INSERT, UPDATE, DELETE.

9.1 Introducción: Lenguaje DML de SQL.

El DML (Data Manipulation Language) es la parte de SQL dedicada a la manipulación de los datos. Las sentencias DML son las siguientes:

- SELECT: se utiliza para realizar consultas y extraer información de la base de datos.
- INSERT: se utiliza para insertar registros en las tablas de la base de datos.
- UPDATE: se utiliza para actualizar los registros de una tabla.
- DELETE: se utiliza para eliminar registros de una tabla.

Hasta ahora hemos estudiado el cómo recuperar datos almacenados en las tablas de nuestra base de datos con las sentencias SELECT. En este tema vamos a tratar 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), después veremos cómo modificar el contenido de las filas de una tabla (con la sentencia UPDATE) y por último cómo borrar filas de una tabla (con la sentencia DELETE). Si trabajamos en un entorno multiusuario, todas estas operaciones se podrán realizar siempre que tengamos los permisos correspondientes.

9.2 La sentencia INSERT.

9.2.1 Sintaxis de la sentencia INSERT.

La sentencia INSERT se utiliza para insertar nuevos registros a una tabla.

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

Inserción de una fila:

Se puede escribir la sentencia INSERT INTO de dos maneras:

- La primera forma no especifica los nombres de las columnas en las que se inserta los datos, sólo se especifican los valores:

```
INSERT INTO table_name VALUES (valor1, valor2, valor3,..);
```

  - En esta opción hay que tener cuidado ya que si no agregamos todos los valores correspondientes a las columnas nos envía un error indicando que hace falta incluir el valor de esa columna que falta.
  - 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 o 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 "autoincrement", lo normal es no asignar valor a esa columna para que el sistema le asigne el valor que le corresponda según el contador (se pone el valor "Null" en la lista de valores), si por el contrario queremos que la columna tenga un valor concreto, lo indicamos en la lista de valores.
  - Los valores se tienen que especificar en el mismo orden en que aparecen las columnas en el diseño de la tabla, y se tiene que utilizar el valor NULL para rellenar las columnas de las cuales no tenemos valores.

- La segunda forma especifica tanto los nombres de las columnas como los valores a insertar:

```
INSERT INTO table_name (columnal, columna2, columna3,...)
                VALUES (valorl, valor2, valor3,...);
```

  - 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 definición de la tabla. Se pueden omitir algunas columnas, la columnas que no se nombran tendrán por defecto el valor NULL o el valor predeterminado indicado en la definición de la tabla.

Inserción de múltiples filas a la vez:

Especificando las columnas que tienen valores concretos:

```
INSERT INTO table_name (columnal, columna2,..)
     VALUES (valor1, valor2,..), (valor1, valor2,..),..
     (valor1, valor2,..);
```

También se puede expresar sin especificar las columnas:

```
INSERT INTO table_name
     VALUES (valorl, valor2,..), (valor1, valor2,..),..
     (valor1, valor2,..);
```

9.2.2 La sentencia INSERT y SELECT.

Con MySQL, podemos copiar información de una tabla a otra.

Podemos insertar en una tabla varias filas con una sola sentencia INSERT INTO si los valores a insertar se pueden obtener como resultado de una consulta. Se sustituye la cláusula "VALUES lista de valores" por una sentencia SELECT que filtre los datos a insertar. Cada

fila resultado de SELECT forma una lista de valores a insertar. Es como si tuviésemos una INSERT ... VALUES por cada fila resultado de la sentencia SELECT.

Sintaxis MySQL:

```
INSERT INTO table2 SELECT * FROM table1;
```

- Cada fila devuelta por la SELECT actúa como la lista de valores que vimos con la INSERT... VALUES. Tienen las mismas restricciones en cuanto a tipo de dato y número de columnas. 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.

También se pueden copiar solo las columnas que nos interesen, indicándolas entre paréntesis detrás de la tabla destino:

```
INSERT INTO table2 (columna_nombre,..)
          SELECT columna_nombre,.. FROM table1;
```

- Las columnas de la sentencia 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 han de coincidir en el número de columnas y en el tipo de dato.

Ejemplo: Supongamos que tenemos una tabla llamada "representantes" con la misma estructura que la tabla "empleados", y queremos insertar en esa tabla los empleados que tengan como puesto "rep ventas"

```
INSERT INTO representantes
      SELECT * FROM empleados WHERE puesto = 'rep ventas';
```

Con la sentencia SELECT obtenemos las filas correspondientes a los empleados con puesto "rep ventas", y las insertamos en la tabla "representantes". Como las tablas tienen la misma estructura no hace falta poner la lista de columnas y se puede emplear “*” en la lista de selección de la sentencia SELECT.

Otro ejemplo: Supongamos ahora que la tabla "representantes" tuviese las siguientes columnas "numemp", "oficinarep", "nombrerep". En este caso no podríamos utilizar el asterisco, tendríamos que poner:

```
INSERT INTO representantes SELECT numemp, oficina, nombre
                FROM empleados WHERE puesto = 'rep ventas';
```

O bien:

```
INSERT INTO representantes (numemp, oficinarep, nombrerep)
           SELECT numemp, oficina, nombre
           FROM empleados WHERE puesto = 'rep ventas';
```

Ejercicio: En la base de datos Universidad, crea una tabla con los siguientes atributos:

- Código: auto incremento.
- Nif.
- Nombre.
- Apellidos.
- Ciudad.
- Dirección.

Debes cargar los datos de la tabla “persona”, pero solo los que son alumnos.

```
mysql> create table alumnos (codigo int(10) auto_increment,
                             nif varchar(9) not null,
                             nombre varchar(25) not null,
                             apellidos varchar(100) not null,
                             ciudad varchar(25),
                             direccion varchar(50),
                             primary key (codigo));
Query OK, 0 rows affected (0.00 sec)


mysql> insert into alumnos
              select null,nif,nombre,concat_ws(" ",apellido1,apellido2),ciudad,direccion from persona
                     where tipo="alumno";
Query OK, 12 rows affected (0.01 sec)
Records: 12  Duplicates: 0  Warnings: 0


mysql> select * from alumnos;
+--------+-----------+----------+----------------------+----------+-------------------------+
| codigo | nif       | nombre   | apellidos            | ciudad   | direccion               |
+--------+-----------+----------+----------------------+----------+-------------------------+
|      1 | 26902806M | Salvador | Sánchez Pérez        | Almería  | C/ Real del barrio alto |
|      2 | 89542419S | Juan     | Saez Vega            | Almería  | C/ Mercurio             |
|      3 | 17105885A | Pedro    | Heller Pagac         | Almería  | C/ Estrella fugaz       |
|      4 | 04233869Y | José     | Koss Bayer           | Almería  | C/ Júpiter              |
|      5 | 97258166K | Ismael   | Strosin Turcotte     | Almería  | C/ Neptuno              |
|      6 | 82842571K | Ramón    | Herzog Tremblay      | Almería  | C/ Urano                |
|      7 | 46900725E | Daniel   | Herman Pacocha       | Almería  | C/ Andarax              |
|      8 | 11578526G | Inma     | Lakin Yundt          | Almería  | C/ Picos de Europa      |
|      9 | 79089577Y | Juan     | Gutiérrez López      | Almería  | C/ Los pinos            |
|     10 | 41491230N | Antonio  | Domínguez Guerrero   | Almería  | C/ Cabo de Gata         |
|     11 | 64753215G | Irene    | Hernández Martínez   | Almería  | C/ Zapillo              |
|     12 | 85135690V | Sonia    | Gea Ruiz             | Almería  | C/ Mercurio             |
+--------+-----------+----------+----------------------+----------+-------------------------+
12 rows in set (0.00 sec)
```

9.3 La sentencia UPDATE.

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

La sintaxis simplificada es la siguiente:

```
UPDATE nombre_tabla SET nombre_columna = nuevo_valor
                        WHERE condición_where
```

- Donde `nombre_tabla` es la tabla a modificar. La cláusula SET especifica qué columna/s van a modificarse y qué valor/es asignar a esa/s columna/s. Separadas por "," si hay más de una.
- `nombre_columna` es el nombre de la columna a la cual queremos asignar un nuevo valor por lo tanto debe ser una columna de la tabla `nombre_tabla`.
- Lógicamente, `nuevo_valor` debe ser un valor del tipo de dato apropiado para la columna indicada.
- La cláusula WHERE indica qué filas van a ser modificadas. CUIDADO: Si se omite la cláusula WHERE se actualizan todas las filas. En la condición del WHERE se puede incluir una subconsulta, pero hay que tener en cuenta que la tabla que aparece en la cláusula FROM de la subconsulta no puede ser la misma que la tabla que aparece como origen (la que se quiere modificar con la sentencia UPDATE) .

Ejemplo: Imaginemos que tenemos la tabla "empleados" que tiene un campo que acumula todas la ventas de cada empleado. A su vez, cada empleado está asignado a una oficina donde trabaja. Queremos poner a cero las ventas de los empleados de la oficina 12:

```
UPDATE empleados SET ventas = 0 WHERE oficina = 12;
```

Otro Ejemplo (usando subconsultas): Queremos poner a cero el límite de crédito de los clientes asignados a empleados de la oficina 12:

```
UPDATE clientes SET limitecredito = 0
                WHERE CodRepVentas IN
                (SELECT numemp FROM empleados WHERE oficina = 12);
```

Nota: Observar que la tabla que aparece en la subconsulta ("empleados"), no es la misma (no puede ser la misma) que la tabla que aparece como origen ("clientes").

Cuando se ejecuta una sentencia UPDATE lo primero que se hace es coger la tabla a modificar seleccionando 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.

9.4 La sentencia DELETE.

La sentencia DELETE elimina filas de una tabla.

La sintaxis simplificada es la siguiente:

```
DELETE FROM nombre_tabla WHERE condición_where
```

- `nombre_tabla` es el nombre de la tabla de donde 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. CUIDADO: Si no se indica la cláusula WHERE, se borran TODAS las filas de la tabla.
- La condición WHERE puede utilizar una subconsulta. En MySQL 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 sentencia DELETE.
- Una vez borrados, los registros no se pueden recuperar.

Ejemplo 1, borra todas las filas de la tabla empleado:

```
DELETE FROM empleado;
```

Ejemplo 2, Elimina las filas de los empleados que pertenezcan al departamento número 5:

```
DELETE FROM empleado WHERE codigo_departamento = 5;
```

Ejemplo 3 (usando subconsultas en el WHERE):

```
DELETE FROM empleado WHERE codigo_departamento =
      (SELECT codigo FROM departamento
                      WHERE nombre = 'Contabilidad');
```

Elimina las filas de los empleados que pertenezcan al departamento de Contabilidad.

Nota 1: Observar que la tabla que aparece en la subconsulta ("departamento"), no es la misma que la tabla que aparece como destino del borrado ("empleado").

Nota 2: Si se intenta de actualizar o borrar registros y MySQL tiene activado el “modo seguro de actualización”, solo nos dejará hacerlo si en la condición trabajamos con la clave principal. En caso contrario no y nos aparecerá el error: 1175.

El modo seguro de actualización se gestiona a través de la variable del sistema: SQL_SAFE_UPDATES.

Para activarla: set SQL_SAFE_UPDATES=1;

Para desactivarla:set SQL_SAFE_UPDATES=0;

9.5 La sentencia TRUNCATE.

La sintaxis es la siguiente:

```
TRUNCATE nombre_tabla;
```

Esto eliminará todos los datos y restablecerá el índice AUTO_INCREMENT. No borra la estructura de la tabla (DDL), solo los datos. Es mucho más rápido que DELETE FROM nombre_tabla cuando hay un gran conjunto de datos. Puede ser muy útil durante el desarrollo / prueba de la BBDD.

En realidad, cuando MySQL "trunca" una tabla, no borra los datos, elimina la tabla y la vuelve a crear.

9.6 Borrado y modificación de datos con integridad referencial.

A la hora de borrar o modificar algún dato o fila de una tabla puede ser que el sistema no nos lo permita, eso es debido a la integridad referencial, visto en el primer trimestre. Cuando se crean tablas relacionadas entre sí a través de las claves foráneas, el sistema se asegura que existan como clave principal en la tabla a la que se relaciona. Si, aún así, se quiere borrar o modificar un dato, habría que modificar la definición de la FOREIGN KEY:

ON DELETE y ON UPDATE: Nos permiten indicar el efecto que provoca el borrado o la actualización de los datos que están referenciados por claves ajenas. Las opciones que podemos especificar son las siguientes:

- RESTRICT: Impide que se puedan actualizar o eliminar las filas que tienen valores referenciados por claves ajenas. Es la opción por defecto en MySQL. Al intentar borrar o actualizar un registro al que se apunta desde otra tabla con una clave foránea dará error 1451.
- CASCADE: Permite actualizar o eliminar las filas que tienen valores referenciados por claves ajenas.
- SET NULL: Asigna el valor NULL a las filas que tienen valores referenciados por claves ajenas.
- NO ACTION: Es una palabra clave del estándar SQL. En MySQL es equivalente a RESTRICT.
- SET DEFAULT: No es posible utilizar esta opción cuando trabajamos con el motor de almacenamiento InnoDB. Puedes encontrar más información en la documentación oficial de MySQL.

9.7 Ejercicios.

9.7.1 Ejercicios sobre base de datos: Tienda de informática.

1. Inserta un nuevo fabricante indicando su código y su nombre.
2. Inserta un nuevo fabricante indicando solamente su nombre.

3. Inserta un nuevo producto asociado a uno de los nuevos fabricantes. La sentencia de inserción debe incluir: código, nombre, precio y código_fabricante.
4. Inserta un nuevo producto asociado a uno de los nuevos fabricantes. La sentencia de inserción debe incluir: nombre, precio y código_fabricante.
5. Crea una nueva tabla con el nombre fabricante_productos que tenga las siguientes columnas: nombre_fabricante, nombre_producto y precio. Una vez creada la tabla inserta todos los registros de la base de datos tienda en esta tabla haciendo uso de única operación de inserción.
6. Elimina el fabricante Asus. ¿Es posible eliminarlo? Si no fuese posible, ¿qué cambios debería realizar para que fuese posible borrarlo?
7. Elimina el fabricante Xiaomi. ¿Es posible eliminarlo? Si no fuese posible, ¿qué cambios debería realizar para que fuese posible borrarlo?
8. Actualiza el código del fabricante Lenovo y asígnale el valor 20. ¿Es posible actualizarlo? Si no fuese posible, ¿qué cambios debería realizar para que fuese actualizarlo?
9. Actualiza el código del fabricante Huawei y asígnale el valor 30. ¿Es posible actualizarlo? Si no fuese posible, ¿qué cambios debería realizar para que fuese actualizarlo?
10. Actualiza el precio de todos los productos sumándole 5 € al precio actual.
11. Elimina todas las impresoras que tienen un precio menor de 200 €.

9.7.2 Ejercicios sobre base de datos: Empleados.

1. Inserta un nuevo departamento indicando su código, nombre y presupuesto.
2. Inserta un nuevo departamento indicando su nombre y presupuesto.
3. Inserta un nuevo departamento indicando su código, nombre, presupuesto y gastos.
4. Inserta un nuevo empleado asociado a uno de los nuevos departamentos. La sentencia de inserción debe incluir: código, nif, nombre, apellido1, apellido2 y codigo_departamento.
5. Inserta un nuevo empleado asociado a uno de los nuevos departamentos. La sentencia de inserción debe incluir: nif, nombre, apellido1, apellido2 y codigo_departamento.
6. Crea una nueva tabla con el nombre departamento_backup que tenga las mismas columnas que la tabla departamento. Una vez creada copia todos las filas de tabla departamento en departamento_backup.
7. Elimina el departamento Proyectos. ¿Es posible eliminarlo? Si no fuese posible, ¿qué cambios debería realizar para que fuese posible borrarlo?
8. Elimina el departamento Desarrollo. ¿Es posible eliminarlo? Si no fuese posible, ¿qué cambios debería realizar para que fuese posible borrarlo?
9. Actualiza el código del departamento Recursos Humanos y asígnale el valor 30. ¿Es posible actualizarlo? Si no fuese posible, ¿qué cambios debería realizar para que fuese actualizarlo?
10. Actualiza el código del departamento Publicidad y asígnale el valor 40. ¿Es posible actualizarlo? Si no fuese posible, ¿qué cambios debería realizar para que fuese actualizarlo?
11. Actualiza el presupuesto de los departamentos sumándole 50000 € al valor del presupuesto actual, solamente a aquellos departamentos que tienen un presupuesto menor que 20000 €.

9.7.3 Ejercicios sobre base de datos: Jardinería.

1. Inserta una nueva oficina en Almería.
2. Inserta un empleado para la oficina de Almería que sea representante de ventas.
3. Inserta un cliente que tenga como representante de ventas el empleado que hemos creado en el paso anterior.
4. Inserte un pedido para el cliente que acabamos de crear, que contenga al menos dos productos.
5. Actualiza el código del cliente que hemos creado en el paso anterior y averigua si hubo cambios en las tablas relacionadas.
6. Borra el cliente y averigua si hubo cambios en las tablas relacionadas.
7. Elimina los clientes que no hayan realizado ningún pedido.
8. Incrementa en un 20% el precio de los productos que no tengan pedidos.
9. Borra los pagos del cliente con menor límite de crédito.
10. Establece a 0 el límite de crédito del cliente que menos unidades pedidas tenga del producto OR-179.
11. Modifica la tabla detalle_pedido para incorporar un campo numérico llamado total_linea y actualiza todos sus registros para calcular su valor con la fórmula:

    ```
    total_linea = precio_unidad*cantidad * (1 + (iva/100));
    ```

12. Borra el cliente que menor límite de crédito tenga. ¿Es posible borrarlo solo con una consulta? ¿Por qué?
13. Inserta una oficina con sede en Granada y tres empleados que sean representantes de ventas.
14. Inserta tres clientes que tengan como representantes de ventas los empleados que hemos creado en el paso anterior.
15. Borra uno de los clientes y comprueba si hubo cambios en las tablas relacionadas. Si no hubo cambios, modifica las tablas necesarias estableciendo la clave foránea con la cláusula ON DELETE CASCADE.