
UD5: MODELO FÍSICO DML

Parte1. Manipulación de Datos

Tema Extendido

Bases de Datos (BD)
CFGs DAM/DAW

Abelardo Martínez y Pau Miñana.
Basado y modificado de Sergio Badal y Raquel Torres.
Curso 2023-2024

ÍNDICE

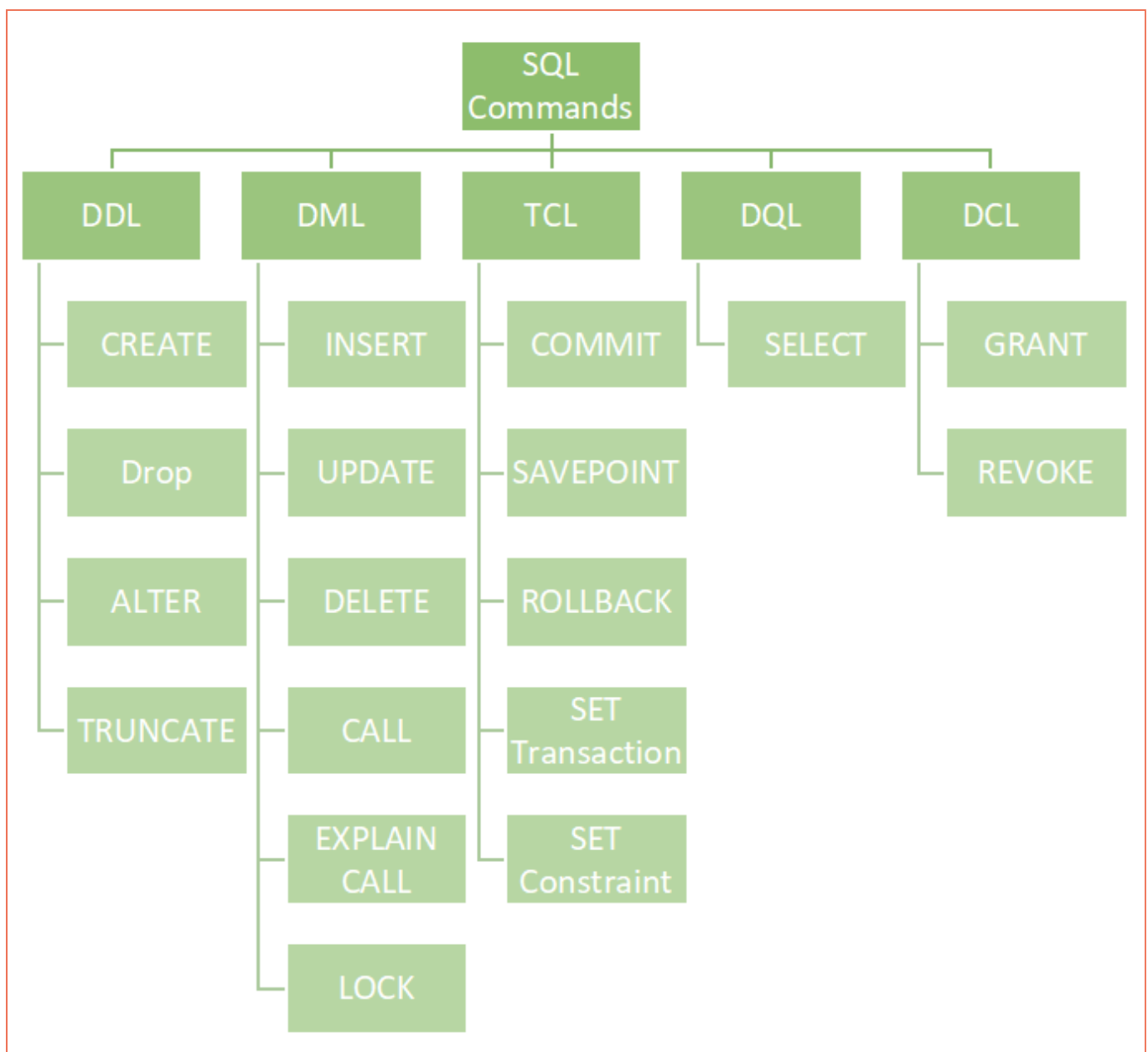
- [1. Introducción](#)
- [2. Inserción de datos](#)
 - [2.1. INSERT INTO](#)
 - [2.2. Lista de campos. Inclusión/no inclusión](#)
 - [2.3. Errores en la inserción](#)
- [3. Modificación de datos](#)
- [4. Borrado de datos](#)
- [5. Bibliografía](#)

1. Introducción

Hasta ahora hemos conseguido crear y modificar los metadatos (DDL); el próximo paso será incluir registros en las tablas que hemos creado.

El **DML** (Data Manipulation Language) lo forman las instrucciones capaces de modificar los datos de las tablas.

Algunos autores incluyen un subgrupo dentro del DML al que llaman TCL (Transaction and Control Language) para tratar las transacciones que veremos la semana que viene. También se incluyen otras como CALL, EXPLAIN o LOCK que no veremos este curso.



2. Inserción de datos

2.1. INSERT INTO

Para insertar un registro utilizaremos la instrucción INSERT cuya sintaxis (una de las varias formas que tiene) es la siguiente:

```
INSERT INTO nombre_tabla [(nombre_col, ...)]  
VALUES ({expresión | DEFAULT | NULL}, ...)
```

En ella podemos distinguir las palabras reservadas INSERT INTO seguidas del nombre de la tabla en la que vamos a guardar los nuevos datos. Opcionalmente podemos poner entre paréntesis los nombres de los campos, de este modo podemos usar el orden que queramos e incluso se pueden omitir algunos, que se pondrán al valor por defecto o NULL. Si no incluimos lista de campos se deberán colocar los valores para todos los campos y en el mismo orden en que fueron creados (es el orden de columnas según las devuelve el comando describe), pues de lo contrario se producirá un error si los tipos no coinciden o se almacenará la información de forma errónea en caso de que los tipos de datos de los campos sean compatibles.

```
INSERT INTO departamentos (cod_dpt, nombre_dpt, ubicacion)  
VALUES ('INF', 'Informática', 'Planta sótano U3');
```

2.2. Lista de campos. Inclusión/no inclusión

La lista de campos a rellenar (cod_dpt, nombre_dpt, ubicacion en el ejemplo anterior) solo es necesaria:

- Si no queremos rellenar todos los campos.
- Si no conocemos el orden de los campos del CREATE TABLE
- Si tenemos la lista de valores de los campos en orden distinto a los del CREATE TABLE

Si no incluimos esa lista de campos, el SGBD entenderá que el orden es el especificado cuando creamos la tabla con CREATE TABLE y **se debe poner un valor para todos ellos**. Igualmente se puede usar NULL o DEFAULT para introducir esos valores

```
INSERT INTO departamentos
VALUES ('INF', 'Informática', NULL);
```

En cualquier caso, siempre podremos cambiar el orden de los mismos con la precaución de cambiar también el orden de los VALUES.

```
INSERT INTO departamentos (nombre_dpt, ubicacion, cod_dpt)
VALUES ('Informática', 'Planta sótano U3' , 'INF');
```

Los campos no rellenados explícitamente con la orden INSERT, se rellenan con su valor por defecto (DEFAULT) o bien con NULL si no se indicó valor alguno.

```
INSERT INTO departamentos (cod_dpt, nombre_dpt)
VALUES ('INF', 'Informática');
```

La ubicación tendrá el valor por defecto, asumiendo que se le diese uno en la creación de la tabla, o NULL en caso contrario. Si tiene restricción de obligatoriedad (NOT NULL) y no valor por defecto, ocurrirá un error.

Por ello, aunque sea un poco más pesado, nuestro consejo es que siempre que realices un INSERT incluyas los nombres de los campos, de tal forma que cuando vayas escribiendo los valores de los campos veas que el orden se corresponde con los campos indicados.

Ejemplo

Veamos cómo funciona con un ejemplo. Recordemos las tablas Departamentos y Empleados que creamos en el tema pasado:

<u>cod_dpt</u>	nombre_dpt*	ubicacion
INF	Informática	Planta sótano U3
ADM	Administración	Planta quinta U2
COM	Comercial	Planta tercera U3
CONT	Contabilidad	Planta quinta U1
ALM	Almacén	Planta baja U1

<u>dni</u>	nombre_emp*	especialidad	fecha_alta	dpt*
12345678A	Alberto Gil	Contable	10/12/2010	CONT
23456789B	Mariano Sanz	Informática	04/10/2011	INF
34567890C	Iván Gómez	Ventas	20/07/2012	COM
45678901D	Ana Silván	Informática	25/11/2012	INF
5678012E	María Cuadrado	Ventas	02/04/2013	COM
67890123A	Roberto Milán	Logística	05/02/2010	ALM

La primera forma será indicando todos los campos y todos los valores (recuerda que si queremos colocar una cadena el valor tendrá que ir entre comillas, da igual simples o dobles).

Para insertar el primer registro de la tabla departamentos haremos:

```
mysql> INSERT INTO departamentos (cod_dpt, nombre_dpt, ubicacion)
-> VALUES ('INF','Informática','Planta sótano U3');
Query OK, 1 row affected (0,00 sec)
```

Para ver si ha funcionado y el registro se ha insertado en la tabla utilizaremos la instrucción SELECT. Esta instrucción, que veremos muy a fondo en el próximo tema, sirve para hacer consultas a la base de datos y mostrar la información que contiene. En este ejemplo vamos a utilizar la consulta más simple, por ejemplo:

```
SELECT * FROM departamentos;
```

Donde:

- SELECT es palabra reservada que indica seleccionar
- El asterisco indica todos los campos de la tabla
- La palabra reservada FROM indica de dónde se van a sacar los datos e irá seguida del nombre de la tabla de donde obtendremos la información.

Es decir, la instrucción selecciona todos los campos de la tabla departamentos y muestra su contenido. El resultado será:

```
mysql> SELECT * FROM departamentos;
+-----+-----+-----+
| cod_dpt | nombre_dpt | ubicacion |
+-----+-----+-----+
| INF     | Informática | Planta sótano U3 |
+-----+-----+-----+
1 row in set (0,00 sec)
```

Vamos a probar insertando el segundo registro de la tabla departamentos sin incluir los nombres de los campos en la instrucción. Además vamos a usar comillas dobles en uno de los campos.

```
mysql> INSERT INTO departamentos
-> VALUES ("ADM",'Administración','Planta quinta U2');
Query OK, 1 row affected (0,01 sec)

mysql> SELECT * FROM departamentos;
+-----+-----+-----+
| cod_dpt | nombre_dpt | ubicacion |
+-----+-----+-----+
| ADM     | Administración | Planta quinta U2 |
| INF     | Informática   | Planta sótano U3 |
+-----+-----+-----+
2 rows in set (0,01 sec)
```

Correcto, ya tenemos dos registros en nuestra tabla departamentos, pero ¿siempre hay que ir uno a uno?

En **MySQL** existe una opción exclusiva del INSERT que nos **permite realizar la inserción de varios registros** en una sola instrucción. Simplemente se pueden añadir tras la palabra VALUES, y separados por comas, tantos paréntesis como se desee. Cada uno conteniendo el valor de un registro a insertar.

```
INSERT INTO departamentos VALUES
('COM'. 'Comercial', 'Planta tercera U3'),
('CONT', 'Contabilidad', 'Planta quinta U2'),
('ALM'. 'Almacén ', 'Planta baja U1');
```

Como puedes observar la sintaxis es prácticamente igual a la anterior con la salvedad que ahora podemos colocar los datos de varios registros entre paréntesis separados por comas. También ahora los nombres de las columnas son opcionales, obviamente.

2.3. Errores en la inserción

A la hora de insertar datos se pueden producir errores por diversas causas, algunos errores que irán en contra de las reglas establecidas mostrarán el error y ya está, pero otros, los más difíciles de encontrar son los que producimos los usuarios y que para la base de datos pueden pasar como correctos.

- **Repetir la inserción de un registro que ya existe.** Como la clave primaria debe ser única debe mostrar un error al intentar insertar un registro con la misma clave. Por ejemplo insertar dos veces el mismo empleado.
 - Atención al uso de códigos/id como claves, esto permite que insertemos sin darnos cuenta registros repetidos siempre y cuando usen un código/id distinto, enmascarando el error.
- Insertar registro(s) con un campo **clave ajena a otro registro que todavía no se ha insertado.** Por ejemplo insertar un empleado asociado a un departamento que no se ha insertado aún.
- Insertar registro(s) con **campos NOT NULL y no indicar el valor** de esos campos.
- Insertar registro(s) con **valores que no corresponden con el tipo de datos** de ese atributo.
- Insertar los datos de los **campos en un orden distinto** al de los mismos.
 - Si el tipo de datos es compatible no salta ningún error, lo que hace más difícil detectarlo. Por eso se recomienda **incluir siempre los nombres de los campos en el INSERT** para tener claro el orden.
- Usar caracteres que **no se correspondan al juego de caracteres y/o colación** especificados.
- Introducir textos que contengan **comillas** simples y/o dobles. Por ejemplo, supongamos al empleado Carles d'Ors. Si intentamos introducirlo como 'Carles d'Ors' la cadena acaba en la "d" y tras la "s" empieza otra.
 - Se puede usar el tipo de comillas contrario para abrir y cerrar la cadena para que funcione ("Carles d'Ors").
 - Si un texto tiene comillas de los 2 tipos se puede usar la barra (\' o \") para poder introducir las comillas ('Carles d'Ors').
 - La barra permite también introducir caracteres especiales como saltos de línea (\n). Para poder introducir la barra como carácter, simplemente se dobla (\\).

3. Modificación de datos


Si en el apartado anterior nos hemos equivocado al insertar los datos, ahora tendremos que modificarlos para dejarlos correctos. Esto implica actualizar con nuevos datos el registro erróneo y para ello SQL nos suministra la instrucción UPDATE. Obviamente también se usa cuando algún dato de las tablas cambie por el motivo que sea. La sintaxis a usar será:

```
UPDATE nombre_tabla
SET columna1={expresión | DEFAULT | NULL} [, columna2...]...
[WHERE condición]
```

La instrucción indica que queremos actualizar (UPDATE) una tabla (nombre_tabla) estableciendo (SET) los campos que queremos modificar a una expresión o valor determinado. Podemos colocar tantos campos con sus nuevos valores separados por comas como necesitemos.

Por último, aunque es opcional, debemos incluir la cláusula WHERE condición que nos permitirá seleccionar sobre qué registros se debe realizar la actualización. Puede ser ninguno, uno, varios o incluso todos los de la tabla, si todos los registros cumplen la condición del WHERE.

Lo habitual es que la condición haga referencia, al menos, a algún campo de la tabla.


 **Mucha atención, si no ponemos el WHERE el cambio se aplicará a todos los registros de la tabla. Un WHERE erróneo también es muy problemático, al afectar registros distintos a los deseados.**

Vamos a aplicar esto sobre la tabla de departamentos, donde por error hemos situado 'Contabilidad' en U2 en vez de U1... ¿Habías notado el error? Por eso es importante ir con sumo cuidado al introducir los datos.

```
UPDATE departamentos
SET ubicacion = 'Planta quinta U1'
WHERE cod_dpt = 'CONT';
```

Aunque las condiciones las veremos con muchísima más profundidad en el próximo tema, tenemos que saber que WHERE cod_dpt = 'CONT' indica que la actualización se aplicará a los registros cuyo código de departamento sea igual a CONT.

Esta condición WHERE se puede aplicar también al SELECT (después del nombre de la tabla en el FROM) para limitar los registros mostrados.

 Es una buena costumbre, sobretodo ahora al principio, **probar en un SELECT primero nuestras condiciones WHERE antes de actualizar o borrar registros** para asegurarnos que los cambios se aplican exactamente en los que queremos. Un error en nuestra condición puede hacer que se vean afectados más campos de los requeridos.

La condición en el WHERE puede contener funciones, atributos de las tablas, literales y operadores de lógicos o de comparación:

Operador	Significado
AND	Devuelve verdadero si las expresiones a su izquierda y derecha son ambas verdaderas
OR	Devuelve verdadero si cualquiera de las expresiones a su izquierda o derecha son verdaderas
NOT	Invierte la lógica de la expresión que está a su derecha, si era verdadera pasa a ser falsa

Operador	Significado
>	Mayor
<	Menor
>=	Mayor o igual
<=	Menor o igual
=	Igual
<> o !=	Distinto

El estándar de SQL para "distinto" es "<>", aunque la mayoría de SGBD acepten "!=" no todos lo hacen.

Por ejemplo, para corregir la ubicación del departamento 'CONT', se podría haber usado:

```
UPDATE departamentos
SET ubicacion = 'Planta quinta U1'
WHERE cod_dpt = 'CONT' OR cod_dpt = 'MKT';
```

Aunque no existe un departamento con código 'MKT', actualizará el de 'CONT'; en caso de existir el otro actualizaría ambos.

4. Borrado de datos

Para eliminar registros utilizaremos la instrucción DELETE de la siguiente forma:

```
DELETE FROM nombre_tabla
[WHERE condición]
```

⚠ ! ¡Recordad que si no ponemos la condición se eliminarán todos los registros de la tabla! Sería equivalente a hacer un TRUNCATE.

Imaginemos que queremos eliminar el registro correspondiente al departamento 'MKT', ejecutaríamos la siguiente instrucción:

```
DELETE FROM departamentos
WHERE cod_dpt = 'MKT';
```

5. Bibliografía

- MySQL 8.0 Reference Manual.
<https://dev.mysql.com/doc/refman/8.0/en/>
- Oracle Database Documentation
<https://docs.oracle.com/en/database/oracle/oracle-database/index.html>
- W3Schools. MySQL Tutorial.
<https://www.w3schools.com/mysql/>
- GURU99. Tutorial de MySQL para principiantes Aprende en 7 días.
<https://guru99.es/sql/>