**SQL (Structured Query Language)**

**Poniendo en práctica la teoría en MySQL: DML**

El lenguaje SQL tiene dos grandes rubros. Ya vimos el corte DDL, ahora nos toca DML.

**DML (Data Manipulation Language)**

A diferencia de **DDL,** que consistía en formar la estructura de la base de datos; es decir, lo que concierne a lo primero: formar nuestras tablas, nuestras relaciones, las restricciones (constrains),... y toda esa parte que tiene que ver con la *estructura* propiamente y que se utiliza muy al inicio de un proyecto, está **DML.** Es DML, a diferencia DDL, quien sí se utiliza mucho a lo largo del proyecto, DDL no (más que todo al inicio). DML trata, no de la estructura de la base de datos; si no, del contenido propio de la base de datos (sus datos). Lo dicho, se trata más bien de la manipulación directa sobre los datos (meterlos, actualizarlos, extraerlos y/o borrarlos); mas no sobre las tablas o, incluso, sobre las bases de datos.

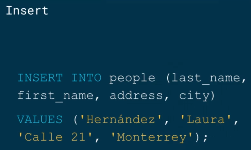
Dicho lo anterior, para ejercer cualquier tipo de acción directa sobre los datos, quiere decir que con anterioridad; además de que ya hay evidencia de una base de datos, la hay de una tabla (al menos); pues, para trabajar con DML y, en consecuencia, con los datos (ya sea meterlos, actualizarlos, extraerlos y/o borrarlos), ya deben existir bases de datos y, dentro de la base de datos en cuestión, tablas (o con una sola basta): lo que es propio a las funciones del lenguaje DDL. Quiere decir entonces que, para trabajar con el lenguaje DML, anteriormente ya debiera haber trabajado con el lenguaje DDL.

**Cuatro comandos principales de DML**



Estos son los comandos que vas a usar, a diferencia de los comandos o sentencias de DDL, a lo largo de todo un proyecto (como por ejemplo *platziblog*): cada vez que quieras meter *(Insert),* actualizar *(update),* borrar *(delete)* o extraer *(select)* una información en concreto (datos). Vamos a ver, una a una, cómo se estructura cada sentencia y, de paso, cómo se operan y se ejecutan ya propiamente dentro de consola MySQL.

**Insert**

, *expliquemos la sentencia* ***Insert*** *parte por parte.*

**1ra parte:**

***Insert into:*** Es la sentencia que tiene como función *insertar* (insertar un dato, o varios) *into,* dentro, de una tabla de una base de datos (la que le pase, estando situados sobre una base de datos*,* y que ya debe estar creada).

***people:*** Es la tabla pasada, de una base de datos, en cuestión; en esta ocasión, la tabla ha sido creada con el nombre de *people;* lo dicho, la tabla ya debe haber sido creada con anterioridad y, dentro de esta sentencia, significa que es la tabla que ha sido seleccionada, *people,* para *insertarle* un nuevo (o nuevos) datos.

***(last\_name, first\_name, address, city):*** son los campos, también atributos (o columnas) ya existentes dentro de una tabla (la tabla de la entidad *people* en este caso), a los que les pasará los nuevos datos que piensa insertar para cada uno de ellos, respectivamente. La cantidad de datos a *insertar* va a depender de la cantidad de parámetros (campos), separados por coma, que usted le pase a la sentencia *insert* dentro de sus paréntesis.

Dato que no pase para una columna o campo en cuestión es rellenada por *default,*... se recuerda este constraint? Es decir, si resulta que no llamó a un campo de su tabla, en la sentencia *insert,...* pasa que dentro de su celda, del nuevo renglón (registro) que se rellene con datos para dicha tabla, se va a poner (recordamos, para la celda de dicho campo que no llamó) un valor por *default* (ya sea el que se pase por defecto, que es NULL, o el que usted le configure con la restricción *DEFAULT*).

**2da parte:**

***values:***Siguiendo el mismo orden respectivo con el que fueron convocados los campos, columnas o atributos,... así mismo, *values,* pondrá justamente los datos que usted desea pasar para dichos campos en cuestión; es decir, en *values*, se evidencia propiamente los valores que usted desea insertar a cada uno de los campos llamados o convocados en la *1ra parte* parte de la sentencia *insert.* Por ejemplo, si teníamos que los campos llamados fueron: ***(last\_name, first\_name, address, city)*** y los valores, o datos, pasados serán: ***(‘Hernandez’, ‘Laura’, ‘Calle 21’, ‘Monterrey’).***

*Quiere decir que,* ***el nuevo dato a insertar en una nueva regla o registro,...***

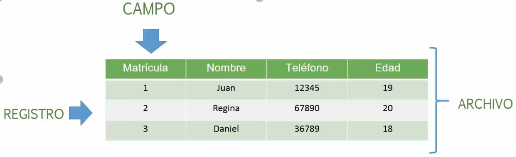
*para* ***last\_name*** *será:* ***‘Hernandez’;***

*para* ***first\_name*** *será:* ***‘Laura’****;*

*para* ***address*** *será:* ***‘Calle 21’****;*

*& para* ***city*** *será:* ***‘Monterrey’...*** *si se da cuenta es muy importante mantener la equivalencia en el orden.*

Tenga en cuenta lo siguiente: **se hace un registro, por renglón, con cada sentencia *value* que hagamos (luego de declarar un *insert*). Es decir, supongamos que los campos, atributos, hacen parte de las columnas de la tabla (la cabecera de ellas); entonces, lo que se hace con *value* es que se pase sólo un nuevo registro, una nueva fila, para insertar sólo un dato para cada una de ellas (para cada una de las columnas o campos).** Recuerde esto:



**Vamos a la consolaahora para crear un nuevo registro (un nuevo renglón o fila con datos para cada campo de una tabla) en nuestra tabla *people* de la base de datos *platziblog.***

Acá toca, por lo general, *codificar* directamente el lenguaje de programación SQL para declarar nuestras sentencias de manipulación directa sobre cada celda o dato de cada campo o columna de una tabla. Esa *codificación* se hace en algo que se llama *terminal* o *cliente gráfico,* tal que así:

Abrimos una nueva pestaña ; ya estando situados y habiendo seleccionado por *default* la base de datos en cuestión, obvio, y la tabla de su interés,... para correr justamente nuestro mismo caso de estudio. Luego, declaramos en la *terminal* o *cliente* *gráfico,* de una nueva pestaña, lo siguiente:

INSERT INTO people (last\_name, first\_name, address, city)

VALUES ('Hernandez', 'Laura', 'Calle 21', 'Monterrey');

***Debe verse así:***



Luego de constatar que, en efecto, queremos ejecutar ese código; que significa, recordemos, agregar un nuevo registro o fila con un nuevo dato para cada una de las columnas o campos de la tabla *people* de la base de datos *platziblog,...* luego de constatar eso, *corremos* .

Y, ***en los mensajes del sistema,*** podrá develar que ya se ejecutó nuestra línea de código y sentencia para afectar nuestra tabla *people. Veamos:*



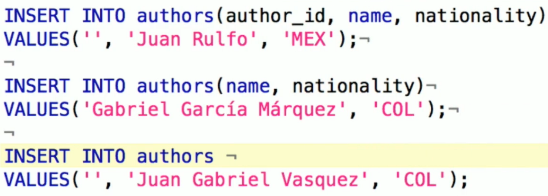
Dice que se afectó*, agregó,* un nuevo registroo fila *(row).* Si todo salió bien, verá un *check* *.* En todo caso, *select* su tabla *people* nuevamente para que visualice que, en efecto, se produjeron los cambios deseados: *insertar* un registro para ella con los datos pasados y en los campos o atributos deseados.

En efecto, todo salió bien:



Si se fija, en el *INSERT INTO people,* dentro de la consola, ni siquiera se consideró manipular directamente al atributo *person\_id.* Pasa que, con el recurso *Auto Increment,* no será necesario manipular dicho campo porque el mismo manejador se encargará de pasarle los enteros uno a uno, como datos, sobre cada registro *insertado* en dicho campo*…* tal como lo hizo en este primer row, *person\_id* fue definido con **1.**

**Pequeño Quiz.**



¿Cuál es la mejor manera de insertar registros de las tres opciones disponibles? **R/** La mejor opción, la que conserva mejor práctica (de hecho, la única de buena práctica) es la del medio. Con las demás la consola arroja *Warnings,* por más que sí ejecuten el código correctamente.

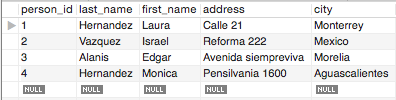
Ahora, puede agregar cuantos ***registros*** desee en una misma sentencia ***value,*** de esta manera: cada conjunto de datos (valores) pasados entre paréntesis ***()*** va a representar un nuevo registro o renglón a la tabla; esto sin necesidad de declarar nuevamente, para cada conjunto de datos por registro o renglón, un ***insert*** o un ***value.* Sólo es necesario separar, por medio de comas, cada conjunto de datos pasados entre paréntesis para una misma sentencia *value.*** Mejor veamos esto en consola:



Si se da cuenta, hemos creado ahora 3 filas más (o registros) a la tabla *people* de la base de datos *platziblog,...*



*y exitosamente.* En efecto, todo sigue saliendo bien:

 Esto es todo en cuanto a ***Insert.***

Se recomienda, como buena práctica, agregar hasta 50 ***registros*** como máximo en una misma sentencia ***value.***

**Update**

La sentencia *Update* tiene que ver con actualizar o modificar los datos, que ya tenemos, dentro de una tabla (archivo, entidad) de una base de datos en especifica; es decir, *update* realmente no inserta nada nuevo a nuestra tabla mientras no sea con la intención de cambiar algún valor o dato ya existente dentro de ella: toma lo que ya hay, en términos de datos, para cambiarlo. Hay diferentes tipos de hacer ese *cambio.* Veamos.

 *Expliquemos sentencia por sentencia*

***Update people:*** Con esto se señala sobre qué tabla haremos las respectivas modificaciones de sus datos ya existentes; en este caso, las modificaciones se harán sobre la tabla *people.*

***Set last\_name:***Está indicando o seleccionando el campo (atributo, columna) de su tabla al que se le pasará un nuevo dato o valor; mas, recuerde, dicho campo con anterioridad ya debe haber ocupado algún valor o dato. Es decir, *last\_name* como campo ya debe de estar ocupando un valor o dato; el cual, deseas cambiar con el nuevo valor o dato que se le pase dentro de la sentencia *SET.*

Qué decimos entonces con la sentencia completa: *SET last\_name = ‘Chavez’, city = ‘Merida’?* Pues, lo que estamos diciendo es que,... se actualice la tabla *people* (previamente señalada con la sentencia: *UPDATE people*)y que efectuemos un cambio en el campo *last\_name* por el valor *‘Chavez’* y, también, que se efectúe un cambio en el campo *city* por el valor de *‘Merida’*. Sin embargo, ambas sentencias, *UPDATE & SET,* son incompletas. Si bien hemos señalado sobre qué tabla efectuar unos cambios (con *update*) y también hemos señalado qué campos de dicha tabla (sus valores) queremos cambiar, no es precisa la información sobre qué renglón o *registro* afectar; es decir, faltaría señalar cuál es el registro o renglón que vamos a estar modificando sus datos o valores. Esto último es, precisamente, lo que se indica con nuestra última sentencia *WHERE.*

***Where person\_id = 1:*** Con esto estamos especificando sobre qué renglón o *registro,* puntual, estaremos efectuando unos cambios de valor o dato para los campos que con anterioridad, por medio de *SET,* han sido indicados. Por lo general en *where* se pasa el valor que asume, dentro del registro, el atributo clave de la tabla en cuestión; es decir, el que tiene la *primary key;* en este caso, ***person\_id = 1***; siendo *person\_id* el atributo clave y *1* su valor (con el que al final es que se logra definir, realmente, qué renglón o registro de la tabla afectar).

 *Veamos todo esto, ahora, en conjunto.*

**El significado de las tres sentencias ejecutadas, conjuntamente, es:** Ve a la tabla *people*, ubícate sobre el registro o renglón que tiene definido como valor *1* para el atributo o campo *person\_id;* y, en ese mismo registro o renglón, cambia los valores de los campos *last\_name* & *city* por los valores (datos) *‘Chavez’* & *‘Merida’*, respectivamente.

Ahora, realmente es una obligación que en *Where* siempre se indique, como campo referencial, al campo que contiene la *primary key?,...* o se podría seleccionar cualquier campo deseado? Por ejemplo, mire este otro caso:



Ahora, vamos más allá, qué pasa cuando no se indica sobre qué registro de la tabla efectuar los cambios; es decir, cuando no se declara una sentencia *Where*? (Peor aún), como es el siguiente caso:



*Veamos todo esto en consola.*

Acá también toca *codificar* directamente el lenguaje de programación SQL para declarar nuestras sentencias de manipulación directa sobre cada celda o dato de cada campo o columna de una tabla. Recuerde, esa *codificación* se hace en algo que se llama *terminal* o *cliente gráfico,* tal que así:

Abrimos una nueva pestaña ; no sin antes, claro, estar ya situados en, y habiendo seleccionado por *default*, la base de datos en cuestión como la tabla de su interés,... todo esto para correr justamente nuestro mismo caso de estudio. Luego, declaramos en la *terminal* o *cliente* *gráfico,* de una nueva pestaña, lo siguiente:

UPDATE people

SET last\_name = ‘Chavez’, city = ‘Merida’

WHERE person\_id = 1;

***Debe verse así:***

 *(fue necesario usar doble comillas en Strings)*

Luego de constatar que, en efecto, queremos ejecutar ese código; que significa, recordemos, cambiar el valor del campo *last\_name* por el valor o dato *‘Chavez’ y* cambiar el valor del campo *city* por el valor o dato *‘Merida’* en el renglón o registro donde *person\_id* tenga por valor definido: *1:* todo esto en la tabla *people* de la base de datos *platziblog,...* luego de constatar lo anterior, *corremos* .

Y, ***en los mensajes del sistema,*** podrá develar que ya se ejecutó nuestra línea de código y sentencia para afectar nuestra tabla *people. Veamos:*



Dice que se afectó*, cambió (changed),* un registroo fila *(row).* Si todo salió bien, verá un *check* *.* En todo caso, *select* su tabla *people* nuevamente para que visualice que, en efecto, se produjeron los cambios deseados: *cambiar* un par de datos, de un registro, por los datos pasados y en los campos deseados.

En efecto, todo salió bien (se hizo los cambios para el registro *person\_id = 1* en los campos *last\_name & city*):



Ahora, recuerda los posteriores dos casos? Estos:

... donde se preguntaba si realmente era una obligación que en *Where* siempre se indicara, como campo referencial, al campo que contiene la *primary key?,...* o si se podría seleccionar cualquier campo deseado? Por ejemplo:



O bien, cuando se preguntaba sobre qué pasaría si no se indicaba sobre qué registro de la tabla desearía efectuar unos cambios; es decir, cuando no se declaraba una sentencia *Where*? Acá:



*Pues bien, pasa que el manejador de bases de datos; en este caso, MySQL, bloquea ambas operaciones, ¿por qué?* Porque es un “*update inseguro”;* estás intentado cambiar masivamente datos sensibles, y muy diversos entre sí, dentro de una tabla (datos que, se supone, contienen su propia identidad) por un valor igual para cada uno de ellos (para cada uno de esos registros que, se creen, son singulares y diferentes… con una propia identidad, lo dicho). Ahora, aclaremos algo, en nuestro primer caso se nos permitió efectuar el cambio debido a que sólo se afectaría un registro: el correspondiente al *person\_id = 1* (no hay más atributos *person\_id* definidos con el valor de *1*)*.* Sin embargo, si tomamos el caso del 1er recuadro de los *updates inseguros*… prácticamente estamos pidiéndole al manejador de la base de datos que, todas las personas que vivan en *Mérida,* se llamen ahora *Juan*; lo cual, en la vida real, no tendría mucho sentido. Con este tipo de sentencias *ambiguas* se lograrían resultados indeseables, por lo cual el manejador de la BD las bloquea.

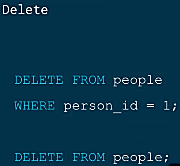
Te imaginas ahora que MySQL nos permita ejecutar el último recuadro? Realmente tendría sentido que todas las personas, no solo las de *Mérida,* pasen a llamarse ahora *Juan?* Evidentemente No! Luego, el mensaje de error que bota el sistema por este tipo de error en especifico es el siguiente:



Igual, tienes la posibilidad de suprimir esta regla... pero no es deseable *(In Preferences).*

**Delete**

Igual que *Drop,* que vimos en lenguaje DDL, ésta es (digamos) su equivalente en DML; ambas muy riesgosas porque *suprimen* información. Concretamente *Delete* se encarga de eliminar o suprimir datos de un *registro* o renglón, señalado, de una tabla en especifica; muy a diferencia de *Drop* que se encarga de eliminar directamente las tablas o, incluso, las mismas bases de datos (y también a diferencia de otra sentencia DDL que es *alter;* pues, una de las tres funciones que tiene, consiste en eliminar o suprimir columnas -mas no tablas o bases de datos). Veamos un poco más sobre *delete.*



***Delete from people:*** Con esto se señala sobre qué tabla haremos las respectivas eliminaciones de sus datos ya existentes; en este caso, las eliminaciones se harán sobre la tabla *people.*

***Where person\_id = 1:*** Con esto estaremos indicando sobre qué renglón o *registro,* puntual, vamos a eliminar sus respectivos valores o datos; es decir, se eliminarían todos los valores o datos de los registros que coincidan con el campo: “person\_ id”y tengan por valor:“1”*.*

Ahora, una declaración todavía más peligrosa: ***Delete from people (sola):*** Lo que hará será eliminar todos los datos de toda la tabla en cuestión, es decir, *people*; pues, no le estamos especificando un parámetro como condición, que deba cumplirse, para ejecutar la tarea de *delete;* yque, anteriormente, lo hacíamos con: *Where person\_id = 1;* (esto especificaba *dónde* querías, puntualmente, borrar tus datos: sobre qué renglón o renglones).

*Veamos todo esto en consola.*

DELETE FROM people

WHERE person\_id = 1;

***Debe verse así:***



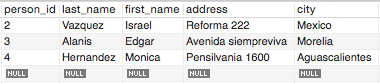
Luego de constatar que, en efecto, queremos ejecutar ese código; que significa, recordemos, borrar los datos del *registro,* o de los *registros,* donde *person\_id* tenga por valor definido: *1:* todo esto en la tabla *people* de la base de datos *platziblog,...* luego de constatar lo anterior, *corremos* .

Y, ***en los mensajes del sistema,*** podrá develar que ya se ejecutó nuestra línea de código y sentencia para afectar nuestra tabla *people. Veamos:*



Dice que se afectó*, eliminó,* un registroo fila *(row)*; en efecto, se eliminó el *registro* donde *person\_id* esigual a *1* de la tabla *people.* Si todo salió bien, verá un *check* *.* En todo caso, *select* su tabla *people* nuevamente para que visualice que, en efecto, se produjeron los cambios deseados: *eliminar* los *registros* que cumplan con la condición *person\_id = 1* de la tabla *people.*

En efecto, todo salió bien.



Ahora, recuerda el posterior caso? Este:



Una sola línea de sentencia que se encargaba de *eliminarnos* todos los datos de la tabla *people...*

*Pues bien, acá también pasa que el manejador de bases de datos; en este caso, MySQL, bloquea la operación, ¿por qué?* Porque es también un “*update inseguro”;* estás intentado eliminar, conjuntamente, todos los datos de una tabla ignorando cualquier tipo de filtro o condición. Por ejemplo, tomemos el caso del recuadro en sí: prácticamente estamos pidiéndole al manejador de la base de datos que elimine todos los datos o valores que conserve la tabla *people.* Realmente, en la vida real, quisiéramos eliminar todos los datos de una tabla que en su momento costó mucho recolectar? Con este tipo de sentencias *peligrosas* se lograrían resultados indeseables, por lo cual el manejador de la BD también las bloquea.



Acá, al igual, tienes la posibilidad de suprimir esta regla... pero no es deseable (In Preferences).

**Select**

*Select* prácticamente es un *filtro,* entiéndalo así, que le permite *seleccionar* o traer información de la base de datos; más específicamente, *seleccionar* los datos de una tabla; esto, a partir de la especificación de unos parámetros o condiciones que le pase, propiamente, a la sentencia *select* en sí para, poder así, visualizar solamente los datos deseados o de su interés de una tabla en cuestión. Por ejemplo, veamos el siguiente caso:



***Select first\_name, last\_name:*** Con esto estamos señalando qué campos o columnas (atributos) deseas ver (sus datos); es decir, queremos ver los datos de los campos o columnas*first\_name, last\_name.*

***From people:*** Luego, en *from,* señalamos de qué tabla (de que entidad de toda la base de datos en su conjunto) queremos ver esos campos; es decir, queremos ver los campos *first\_name, last\_name* de la tabla *people.*

Importante. Ahora, en caso que quiera que su filtro de búsqueda sea más exigente o más especifico sobre lo qué desea realmente ver de la tabla, puede declarar también una sentencia *where;* esto con el motivo de ver, solamente, los datos, no de un campo o columna completa; sino, de un o unos registros en cuestión (y no todos) de dichos campos o columnas indicadas. Si no pasamos un parámetro o condición para *where;* se mostrarán, entonces, todos los renglones o *registros* de la tabla en análisis para los campos o columnas seleccionadas; es decir, en este caso, se verían todos los *registros* de*first\_name & last\_name.*

*Veamos todo esto en consola.*

SELECT first\_name, last\_name

FROM people;

***Debe verse así:***



Luego de constatar que, en efecto, queremos ejecutar ese código; que significa, recordemos, seleccionar los datos (para visualizar) de los campos o columnas *first\_name* & *last\_name* de la tabla *people* de la base de datos *platziblog,...* luego de constatar lo anterior, *corremos* .

Y, ***en los mensajes del sistema,*** podrá develar que ya se ejecutó nuestra línea de código y sentencia para afectar nuestra tabla *people. Veamos:*

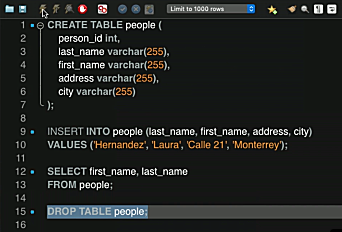


Dice que se afectó*, seleccionó* (select)*,* todos los registroso filas *(rows)* de los campos o columnas *first\_name* & *last\_name*. Si todo salió bien, verá un *check* *.*

En efecto, todo salió bien.



**ALGO IMPORTANTE:** Si usted desea puede, bajo un mismo cliente gráfico o terminal, escribir varias sentencias (diferentes entre sí) que no necesariamente quiera correr o ejecutar al mismo tiempo (ya sean sentencias propias al lenguaje DDL o DML); es decir, en caso tal lo deseado sea ejecutar una de esas tantas sentencias que estén ahí escrita; usted debe seleccionar, dicha sentencia, y arrastrar el mouse sobre ella,... para luego sí correr; tal que así, por ejemplo:

 *si ve que sólo se correrá una sentencia?*

Bajo este pequeño ejemplo, aquí se pone en evidencia también que, dentro del lenguaje SQL, las sentencias propias a DDL se ejecutan sólo al inicio de la estructuración de un proyecto y al final; mientras que, las sentencias relacionadas a DML, son las que se corren en el transcurso o desarrollo del proyecto mismo.

El lenguaje SQL es un lenguaje universal; por lo que, las sentencias se ejecutan de la misma manera, exactamente, en cualquier manejador de bases de datos (no como en un lenguaje de programación que para correr el mismo programa, pero en otro lenguaje, toca cambiar toda una sintaxis); es decir, pongamos esto en contexto, en caso tal usted desee ejecutar las mismas sentencias que corrimos en el cliente gráfico de MySQL en, por ejemplo, Google Cloud Platform,... sólo necesitaría copiar y pegar esas mismas declaraciones de MySQL en la terminal del manejador de bases de datos de Google, y listo. Presione *enter* y a correr!

Algo para agregar de Google Cloud Platform: para verificar, dentro de nuestra terminal, cuáles son las tablas que tenemos el comando que se utiliza es: \***dt*** *(backslash* *+ dt)*

Hay diferencias, como el uso de \***d*** en GCP, mas, como se percata, son diferencias que más bien tienen que ver con el funcionamiento interno de cada manejador y no, propiamente, con las sentencias de los lenguajes DDL (de estructura) y DML (de contenido) de SQL (esto es igual para cualquier manejador de DB con estándar SQL).

**RECUERDE**: Cada tabla creada, de una base de datos, realmente representa una **entidad** independiente de todas las existentes para dicha base de datos **(una tabla NO representa un atributo, campo o columna; la tabla en sí contiene varios atributos, campos o columnas).**