APUNTES SQL

José Juan Urrutia Milán

Reseñas

- Curso SQL: https://www.youtube.com/watch?v=Bk3rY_ICgPo&list=PLU8oAlHdN5Bmx-LChV4K3MbHrpZKefNwn&index=2
- Instalación del software (min 5:55 y vídeo 3 del mismo curso): https://www.youtube.com/watch?v=tXxOAXP-gkg&list=PLU8oAlHdN5BkinrODGXToK9oPAlnJxmW_&index=2
- Uso PHPmyadmin: https://www.youtube.com/watch?v=LKrBvE3yAZo&list=PLU8oAlHdN5BkinrODGXToK9oPAlnJxmW_&index=33

Siglas/Vocabulario

- **IDE:** Entorno de Desarrollo Integrado.
- **BBDD:** Bases de Datos.
- **SGBD:** Sistemas Gestores de Bases de Datos.

Leyenda

Cualquier abreviatura o referencia será subrayada.
Cualquier ejemplo será escrito en **negrita.**
Cualquier palabra de la que se pueda prescindir irá escrita en cursiva.
Cualquier abreviatura viene explicada a continuación:

Abreviaturas/Referencias:

- 123: Hace referencia a cualquier número.
- nombre: Hace referencia a cualquier palabra/cadena de caracteres.
- a: Hace referencia a cualquier caracter.
- cosa: Hace referencia a cualquier número/palabra/cadena.
- Tipo_var o **…** : Hace referencia a cualquier tipo de variable primitiva o de tipo String.
- nombre_var: Hace referencia a cualquier nombre que se le puede dar a una variable.
- código: Hace referencia a cualquier instrucción. (Se usará para indicar dónde se podrá inscribir código.)
- variable: Hace referencia a cualquier variable.
- condición: Hace referencia a cualquier condición. Entiéndase por condición, una afirmación que devuelve un true o un false. Ej: (variable == 123). *Una variable del tipo boolean puede ser usada como una condición.

Índice

Introducción

Curso destinado al aprendizaje del lenguaje SQL o sequel (lo mismo) (Structured Query Language (Lenguaje estructurado de consultas)).
Para este curso, necesitaremos un SGBD.
SQL no es considerado un lenguaje de programación.
Todos los SGBD tienen un estándar de SQL pero cada uno incorpora nuevas versiones propias de este.
A lo largo de este curso, usaremos el SGBD MySQL (reseñas para instalarlo).
SQL no es case sensitive, podemos escribir sentencias en mayúsculas y en minúsculas.
El signo decimal en SQL es el punto “.” .

Capítulo I: Conceptos básicos

Título I: Tipos de comandos

DDL

Data Definition Language, usados para crear y modificar la estructura de una BBDD. Con él podemos crear tablas, eliminar tablas y modificar la estructura de las tablas, como por ejemplo agregando y quitando campos.

DML

Data Manipulation Language, usados para seleccionar registros de una BBDD (consultas), insertar nuevos registros, borrar información… Se utilizan para hacer consultas de selección (modifican) y de acción (no modifican).

DCL

Data Control Language, usados para proporcionar seguridad en la BBDD.

TCL

Transaction Control Language, usados para la gestión de los cambios en los datos.

Título II: Comandos y cláusulas

Comandos

DDL: create, alter, drop, truncate
DML: select, insert, update, delete
DCL: grant, revoke
TCL: commit, rollback, savepoint

Cláusulas

from

Especifica la tabla de la que se quieren obtener los registros.

where

Especifica las condiciones o criterios de los registros seleccionados.

group by

Para agrupar los registros seleccionados en función de un campo.

having

Especifica las condiciones o criterios que deben cumplir los grupos.

order by

Ordena los registros seleccionados en función de un campo.

limit

Limita el número de resultados de una consulta.

Operadores

<	Menor que.
>	Mayor que.
=	Igual que.
>=	Mayor o igual que.
<=	Menor o igual que.
<>	Distinto que.
between	Entre, utilizado para especificar rangos de valores.
like	Cómo, utilizado con caracteres comodín.
in	En, para especificar registros en un campo concreto.
and	Y lógico.
or	O lógico.
not	Negación lógica.
all	Hace referencia a todos en un grupo.
any	Hace referencia a cualquiera en un grupo.

distinct Se escribe delante de un campo. Hace que no aparezcan datos repetidos de ese campo.
distinctrow Se escribe delante de los campos seleccionados. Hace que no aparezcan registros idénticos en la consulta, mostrando solo 1.

Funciones de agregado

avg()

Calcula el promedio de todos los elementos de un campo.

count()

Cuenta los registros de todos los elementos de un campo.

sum()

Suma los valores de un campo.

max()

Devuelve el valor máximo de un campo.

min()

Devuelve el valor mínimo de un campo.

current_user()

Devuelve el usuario que acaba de modificar la BBDD (útil con los Triggers).

now()

Devuelve la fecha y hora actual

Capítulo II: Comandos DML

Data Manipulation Language, usados para seleccionar registros de una BBDD (consultas), insertar nuevos registros, borrar información… Se utilizan para hacer consultas de selección (no modifican) y de acción (modifican).

Título I: Instrucciones SQL

Una instrucción SQL consta de un comando, unas cláusulas, unos operadores y unas funciones (aunque no es necesario que tenga todas las partes, todo depende de la situación):

Comando + Cláusulas + Operadores + Funciones

*Toda instrucción debe tener al menos un comando y una cláusula.

Orden de cláusulas

Comando + from + where + group by + having + order by

*No es necesario usarlas todas, depende de la situación.

Título II: Selección de campos: Cláusula from

select nombrecampo from nombretabla

Se pueden seleccionar los campos deseados, separarlos por comas.
Si queremos seleccionar todos los campos de la tabla:
select * from nombretabla

Título III: where

Podemos especificar que sólo obtendremos aquellos campos que tengan una condición en específica. Por ejemplo, queremos seleccionar todos los campos donde el campo ID sea menor que 5:
select * from nombretabla where ID<5

Si queremos seleccionar el nombre y el ID de una tabla donde el material del objeto sea plástico:
select nombre, ID from nombretabla where material=”plástico”

Si queremos seleccionar el ID de una tabla donde el material del objeto sea plástico o madera:
select ID from nombretabla where material=”plástico” or material=”madera”

Si queremos seleccionar todos los campos de una tabla donde las IDs estén entre 3 y 10:
select * from nombretabla where ID between 3 and 10

like

Like nos permite seleccionar campos que empiecen o terminen por ciertas letras (o palabras) (no discrimina mayúsculas):
select campo from nombretabla where id like ‘1%’
select campo from nombretabla where id like ‘%89’
select campo from nombretabla where nombre like ‘%balon%’
El primer ejemplo nos da el campo correspondiente de la tabla correspondiente cuya id empiece por 1.
El segundo ejemplo nos da el campo correspondiente cuya id termine por 89.
El tercer ejemplo nos da el campo correspondiente de la tabla cuyo nombre contenga la palabra balon (al principio, en medio y final).

***%** sustituye a cadenas de caracteres y **_** sustituye a caracteres , por lo que si queremos mostrar todos los resultados que tengan que ver con niños y niñas:
select * from tabla where campo like ‘niñ_s’;

Título IV: order by

Order by nos permite ordenar los campos seleccionados en función de los valores de ciertos campos.
Si introducimos un campo de texto como criterio a ordenar, ordenará los campos alfabéticamente; si introducimos una variable numérica, ordenará los campos de menor a mayor.
Para invertir esto (de la Z-A y de mayor a menor), añadimos **desc** al final de la cláusula order by:
Seleccionar todo de una tabla en función de su ID:
-De menor a mayor:
select * from nombretabla order by ID
-De mayor a menor:
select * from nombretabla order by ID desc

Se pueden especificar diferentes campos para ordenar, para en el caso de que varios elementos coincidan en el primer campo, se ordenen según el segundo:
select * from nombretabla order by material, precio
En este caso, si dos artículos están hechos del mismo material, el de menor precio aparecerá antes que el de mayor precio.

Título V: group by

Si queremos obtener la suma de precios de todos los productos de una tabla en función de la sección a la que pertenecen:
select seccion, sum(precio) from nombretabla group by seccion

*Si queremos ordenar la consulta anterior en función de los precios, como no hacemos referencia al precio de ningún producto si no al precio de toda la sección, deberemos establecer un alias con la cláusula **as**:
select seccion, sum(precio) as precios from nombretabla group by seccion order by precios

Si queremos establecer criterios en un grupo, en vez de la cláusula **where**, debemos usar la cláusula **having**, que funciona igual pero se especifica después del **group by**.

Si queremos ver cuantos elementos hay de cada sección:
select count(elementos), seccion from nombretabla group by seccion

Título VI: Consultas de cálculo o de campo nuevo calculado

Las consultas de cálculo se usan para realizar cambios, como operaciones matemáticas, a consultas.
Si queremos sumar a todos los elementos de un campo el mismo número podemos hacer lo siguiente:
select precio+10 from nombretabla .

*Dentro de una función no puedes utilizar el alias dado en esa misma consulta:
select nombre, now() as dia, datediff(dia, fecha) as dif from nom
No funciona, ya que dentro de datediff le estamos especificando el alias dia; deberemos hacer lo siguiente para que el código funcione:
select nombre, now() as dia, datediff(now(), fecha) as def from nom .

round() / truncate()

Las dos se usan de la misma forma:
Especificamos dentro de los paréntesis la función a redondear o truncar y posteriormente y separados por una coma, el número de decimales con el que nos quedamos.
De esta forma, si queremos pasar 1.545454 a 1.54; deberemos usar la siguiente expresión:
ROUND(1.545454, 2)

*Round() y Truncate() se suelen expresar con campos y no con dígitos aislados: **select round(precio, 3) as p from nombretabla** .

now()

Devuelve el día actual. Este se puede implementar como un nuevo campo:
select nombre, precio, now() as dia from nombretabla .

datediff()

Devuelve la diferencia que hay entre dos fechas.
Recibe dos parámetros; la fecha más reciente y la fecha más antigua.
Devuelve los días de diferencia.

date_format()

Establece el formato de la fecha:
Recibe dos parámetros, la fecha a cambiar y el formato que este campo tendrá;
Si queremos que se muestre el día y el mes, en formato especificaremos: **'%D-%M'**

Título VII: Consultas multitabla

Consultas de unión externa

union

Para usar union, ambas tablas deben tener el mismo número de campos y los campos deben tener tipos de datos compatibles (moneda y numérico sí; numérico y texto no).
Union une ambas tablas, haciendo que la segunda se ponga posteriormente a la primera y usando el nombre de los campos de esta.
Si hay registros idénticos en las dos tablas, los une en uno.

Queremos obtener los datos de 2 tablas donde el campo SECCION de la primera sea DEPORTES y de la segunda DEPORTES_EXTREMOS:
select * from tabla1 where seccion=’deportes’ union select * from tabla2 where seccion=’deportes_extremos’

*Se pueden añadir los union que se desee.

union all

Si hay registros idénticos en las dos tablas, no los une.

Consultas de unión interna

Para realizar consultas de unión interna, las tablas que vayamos a consultar deben estar relacionadas entre sí.
Los Joins o consultas de unión interna, unen los dos campos en horizontal, especificando primero los campos de la primera tabla y posteriormente, los de la segunda.

INNER JOIN

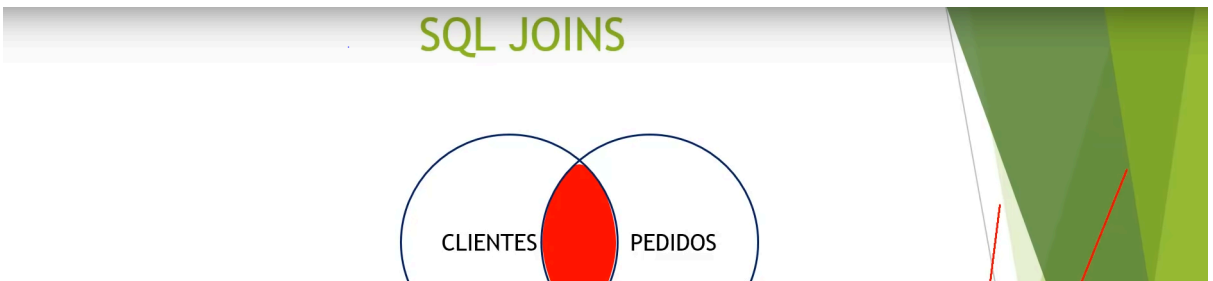
Este Join nos devuelve la información común que hay en ambas tablas. Nos devuelve los registros que hay en ambas tablas.

LEFT JOIN

Este Join nos devuelve la información que hay en ambas tablas y la información que hay en la primera tabla, en este caso la información de la segunda se muestra vacía.

RIGHT JOIN

Este Join nos devuelve la información que hay en ambas tablas y la información que hay en la segunda tabla, en este caso la información de la primera se muestra vacía.



select * from clientes inner join / left join / right join pedidos on clientes.codigocliente=pedidos.codigocliente where pueblo='Madrid'

Título VIII: Subconsultas

En las consultas normalmente aparecerán dos select: Uno externo o padre y uno interno o hijo.
Las subconsultas se especifican entre paréntesis.

Subconsultas escalonadas

Donde el select hijo se mostrará en una única columna. Estas consultas se suelen realizar cuando queremos mostrar por ejemplo, todos los campos de aquellos registros cuyo precio están por encima de la media. La media la obtenemos con una subconsulta.
select nombre, seccion from tabla where precio>(select avg(precio) from tabla)

Subconsultas de lista

Nos devuelve una lista de registros en vez de un único registro.

Subconsultas IN/NOT IN

Estas subconsultas nos permiten hacer que se muestren los datos donde un campo esté (IN) o no esté (NOT IN) en otra subconsulta.

Título IX: Índices

Esta parte del curso fue saltada debido a que no pertenece al estándar SQL y a que sirven para manejar gran cantidad de registros, cosa que a priori me es innecesaria.
(Consultar índices: vídeo 18 a 19).

Título X: Vistas / Views

Las vistas nos permiten crear "accesos directos" a consultas que nos disparen siempre la misma consulta, haciendo que el motor de la BBDD no tenga que trabajar. Estas vistas se actualizan si actualizamos la información de la tabla padre.
Las vistas son útiles para sentencias de consultas que se suelen hacer en repetidas ocasiones de forma frecuente.

Crear una vista

create view <u>nombre</u> as
<u>sentencia</u>
ej:
create view articulos_deportes as
select nombre, unidades, precio from productos where seccion='deportes'

Modificar una vista

alter view <u>nombre</u> as
<u>sentencia</u>
ej:
alter view articulos_deportes as
select nombre, precio from productos where seccion='deportes'

Eliminar una vista

drop view nombre
drop view articulos_deportes

Título XI: limit

limit nos permite seleccionar cuantos resultados queremos ver de cierta consulta. limit establece un número para todos los resultados de la consulta, empezando a contar desde 0.
Para especificar un **limit** , tenemos que es pecificar desde qué elemento empezará la consulta y cuantos elementos queremos extraer:

select nombre from deportes where tipo=’grupo’
Esta sentencia muestra 9 resultados:
fútbol, baloncesto, balonmano, béisbol, hockey, curling, rubgi, fútbol americano.

select nombre from deportes where tipo=’grupo’ limit 2,4
Esta sentencia muestra 4 resultados, empezando a extraerlos desde el resutlado 2 (el 3o):
balonmano, béisbol, hockey, curling.

Capítulo III: Consultas de acción

Las consultas de acción nos permiten actualizar tablas, crear tablas, eliminar tablas y anexar datos a tablas ya existentes mediante comandos DML y DDL:
create, update, delete, insert into y select into.

Título I: Consultas de actualización

Nos permiten modificar datos ya existentes en tablas con la ayuda del comando **update**.
Si queremos aumentar en 10 el precio de todos los productos que pertenezcan a la sección deportes:
update tabla set precio=precio+10 where seccion='deportes'

Título II: Consultas de creación de tablas

Nos permiten crear tablas a partir de otras con la ayuda del comando **create table**.
Si queremos crear una tabla nueva que contenga todos los campos de otra tabla donde el municipio sea igual a Madrid:
create table Clientes_Madrid select * from tabla where municipio='Madrid'
*Clientes_Madrid es el nombre de la nueva tabla y este se puede sustituir por el que queramos.

Título III: Consultas de eliminación

Nos permiten eliminar registros, con la ayuda del comando **delete**.
Para eliminar todos los objetos cuyo municipio sea Madrid:
delete from tabla where municipio='Madrid' .

Título IV: Consultas de datos anexados

Nos permiten anexar todos o parte de los registros de una tabla a otra. Para que esto se pueda realizar, ambas tablas deben tener el mismo número de campos y estos deben de ser de un formato similar.
Usaremos el comando **insert into** , acompañado de un **select**.
Para añadir dentro de la tabla clientes la información entera de la tabla clientes_madrid:
insert into clientes select * from clientes_madrid

Para añadir dentro de clientes algunos campos de clientes_madrid:
insert into clientes (codigo, empresa, telefono) select codigo, empresa, telefono from clientes_madrid

*Al anexar datos, estos datos se colocan al final de la tabla.

Capítulo IV: Uso de comandos DDL

Data Definition Language, usados para crear y modificar la estructura de una BBDD. Con él podemos crear tablas, eliminar tablas y modificar la estructura de las tablas, como por ejemplo agregando y quitando campos.
Los comandos principales son:

create Para crear BBDD y tablas.
alter Para modificar las propiedades de las tablas.
drop Para eliminar BBDD, tablas, campos y propiedades.
truncate Para borrar todas las filas de una tabla sin borrar la tabla.

Título I: Creación de tablas

Para crear una nueva tabla, ejecutaremos la siguiente sentencia:
create table nombretabla (campo1 tipodato1, campo2 tipodato2, campo3, tipodato3...)
ejs:
create table prueba (nombre varchat(20), edad tinyint, fecha date, masculino bool)
*tinyint permite almacenar números enteros entre el 0 y el 255 (4 dígitos).
**El dígito entre los paréntesis de varchat especifica el número máximo de caracteres que esta cadena tendrá.

Establecer campo autonumérico (para ids)

Para ello, deberemos además especificar que es un campo clave. Con la función primary key() al final de todos los campos:
Para crear un campo autonumérico, agregaremos auto_increment después de indicar que es de tipo int.
create table prueba (id integer auto_increment, nombre varchat(20), primary key(id))

o
create table prueba (id integer primary key autoincrement, nombre varchat(20))

Establecer keys no autonumericas

Si queremos que nuestra key no sea autonumerica (porque por ejemplo, es un string (AR01, AR02...)), simplemente indicamos **primary key** después del tipo de valor:
create tabla prueba(id varchat(4) primary key, nombre varchat(20))

Establecer campo único

Si queremos implementar un campo en el que la información no se pueda repetir, pero que este no sea la key del registro (como por ejemplo, un campo para DNIs), debemos crear un campo único:
create table prueba (id integer primary key autoincrement, nombre varchat(20), dni varchat(9) unique)
Si intentamos crear dos registros diferentes con el mismo valor en el campo dni, nos dará error.
Pueden haber tantos campos únicos como se desee.

Tipos de datos

- Cadena: Cadena de texto.
- Numérico: Un número.
- Fecha/Hora: Muestra datos sobre el tiempo.
- Booleanos: True o false.

(PDF con todos los subtipos de datos (y los posibles nombres de estos): https://www.youtube.com/watch?v=XJb6qflbsx4&list=PLU8oAlHdN5Bmx-LChV4K3MbHrpZKefNwn&index=16 ver descripción de vídeo).

Título II: Eliminación de tablas

Para ello, seguiremos el siguiente esquema:

drop table nombretabla

ej:

drop table prueba

Título III: Agregar y borrar campos de tablas

Agregar campos

alter table nombretabla add column nombrecampo tipodato

ej:

alter table prueba add column municipio varchat(20)

Borrar campos

alter table nombretabla drop column nombrecampo

ej:

alter table prueba drop column municipio

Título IV: Modificar características de campos

Seguimos el siguiente esquema:

alter table nombretabla alter column nombrecampo tipodato

ej:

alter table prueba alter column municipio date

Título V: Establecer valor por defecto de un campo y eliminar valor por defecto de un campo

Establecer valor por defecto

alter table nombretabla alter column nombrecampo set default 'texto'

ej:

alter table prueba alter column municipio set default 'Madrid'

Eliminar valor por defecto

alter table nombretabla alter column nombrecampo drop default

ej:

alter table prueba alter column municipio drop default

Título VI: Crear nuevos registros

insert into nombretabla (campo1, campo2...) values(valor1, valor2...)

*Si se van a incluir valores para todos los campos de la tabla, se puede prescindir de los paréntesis en los que le indicamos los campos:

insert into nombretabla values(valor1, valor2, valor3)

Título VII: Actualizar registros

update nombretabla set campo=valor

*Se puede insertar un **where** para que no todos los registros se vean afectados:

update productos set precio=precio+10 where codigo=1

Título VIII: Eliminar registros

delete from nombretabla where campo=valor

El where lo especificamos para no borrar la tabla entera, sólo un registro en el que un determinado campo tiene un determinado valor.

Título IX: Reordenar tabla después de eliminar

MySQL

Tenemos una tabla con un campo id primary key autoincrement. Después de eliminar algunos registros queremos volver a reordenar la tabla:

ALTER TABLE tablename DROP id;

ALTER TABLE tablename AUTO_INCREMENT = 1,

ALTER TABLE tablename ADD ID INT AUTO_INCREMENT PRIMARY KEY FIRST;

SQLite

Tenemos una tabla con un campo id primary key autoincrement y name text.

Después de eliminar algunos registros queremos volver a reordenar la tabla:

CREATE TABLE t1_backup (ID INTEGER PRIMARY KEY AUTOINCREMENT, NAME TEXT);

INSERT INTO t1_backup (NAME) SELECT NAME FROM PRUEBA1;

DROP TABLE PRUEBA1;

ALTER TABLE t1_backup RENAME TO PRUEBA1;

Capítulo V: Triggers y procedimientos almacenados

Definiciones

Un Trigger es un objeto asociado a una tabla que dispara un evento cuando sucede algo en esta (antes o después de insertar registros, antes o después de actualizar registros, o antes o después de eliminar registros).
Los Triggers nos permiten realizar cosas automáticas como llevar un registro de los usuarios que insertan registros (guardar el nombre de usuario (consultar Capítulo I, Título II, **Funciones de agregado**), el día, la hora y la ID del registro que ha cambiado), realizar copias de seguridad en otra tabla cuando se eliminan datos…
Los Triggers sirven para realizar tareas de mantenimiento y administración en una BBDD, aunque también nos permite controlar algunos campos, como por ejemplo que un campo llamado edad no pueda tener un valor negativo.

Los procedimientos almacenados son sentencias SQL que podemos almacenar en nuestra BBDD para que, si tenemos diferentes aplicaciones que suelen repetir cierta sentencia, en vez de repetir siempre la sentencia, podamos llamar a una "función" desde el programa que tendrá los mismos efectos que esta sentencia.
Podemos entender que los procedimientos almacenados son una especie de funciones en las BBDD.

Título I: Creación de Triggers

Para crear un Trigger, deberemos seguir el siguiente esquema:
1. Crear el trigger y asignarle un nombre.
2. Decirle al trigger si va a realizar la acción antes o después (before/after).

3. Decirle al trigger cuando se va a disparar (al insertar información (insert), actualizar (update) o borrar (delete) y el nombre de la tabla con la que se asocia).
4. Determinar si la acción se va a disparar cuando se realice la acción por cada fila (for each row) o por cada sentencia (for each statement)
5. Determinar qué va a hacer el trigger: insertar en alguna tabla, actualizar alguna información, borrar información…
Para esto, nos podemos ayudar de los parámetros new y old.
- new nos permite hacer referencia al valor de un campo introducido en la tabla a la que el trigger pertenece, ej:
create trigger valores_ai after insert on valores for each statement insert into tabla(precio) values(new.precio)
así, insertamos en el campo **precio** de la tabla **tabla** el nuevo valor del campo **precio** de la tabla **valores** (el campo origen y destinatario no tienen porqué llamarse igual como en el ejemplo).
- old nos permite acceder al valor antiguo que había en un campo. Este valor fue cambiado y podemos recuperar el antiguo con old.
Old se usa de la misma forma que new.
Para que podamos usar old, el trigger debe ejecutarse antes (before)

create trigger nombre before/after insert/update/delete on tablatrigger for each row/statement insert into tabla(campo1, campo2...) values(new.dato1, new.dato2...)

Convención nominal

Los Triggers se suelen llamar de una cierta forma por convención entre programadores:
Primero, el nombre de la tabla a la que se asocia.
Después una barra baja y una letra dependiendo si se va a ejecutar antes (B) o después (A) de hacer la acción.
Posteriormente, una letra haciendo referencia a la acción que va a desencadenar el trigger: insertar (I), actualizar (U) o borrar (D).
ej: productos_ai

Título II: Modificar Triggers

Para ello:
Deberemos borrar el trigger y volverlo a crear (estas dos cosas se pueden realizar en una única sentencia), ya que no se puede modificar un trigger.
drop trigger if exists nombretrigger; create trigger 'nombretrigger' after/before insert/update/delete on nombretabla for each row/statement insert/update/delete into nombretabla2 …

*Los triggers se pueden modificar manualmente desde MySQL gracias a su interfaz gráfica.

Título III: Procedimientos almacenados

*Si el procedimiento almacenado presenta más de una línea, es necesario indicar donde empieza (begin) y donde acaba (end).
(ver título IV).

Crear un procedimiento almacenado

create procedure nombre() sentencia

*La sentencia puede ser cualquiera de las vistas en este manual.
ej: (queremos que nuestra sentencia nos devuelva todos los campos de todos los registros de una tabla llamada productos donde su localización sea Madrid)

```
create procedure obtener_madrid()
select * from productos where localizacion='Madrid'
```

Procedimiento almacenado que recibe parámetros

create procedure nombre(nombre tipodato, nombre tipodato, ...) sentencia

*Para utilizar parámetros, utilizamos su nombre como si de un lenguaje de programación se tratara:
create procedure actualizar_precio(n_precio int, n_codigo int)
update productos set precio = n_precio where codigo = n_codigo

Llamar a un procedimiento almacenado

call nombre()
ej: call obtener_madrid()
***Si este recibe parámetros, indicar estos parámetros (recordando que las cadenas de texto tienen que estar entre ‘’):
call actualizar_precio(600, 410)

Título IV: Procedimientos almacenados con variables

Vamos a crear un procedimiento almacenado que nos devuelva nuestra edad si le indicamos el año de nacimiento. A priori esto no tiene nada que ver con una BBDD pero sí que puede ser de utilidad a la hora de entender los procedimientos almacenados con variables:

Para declarar una variable, usaremos la palabra **declare**, seguido del nombre que queramos darle a la variable. Posteriormente, indicamos de qué tipos será la variable.
Si queremos darle un valor para inicializarla, indicaremos la palabra **default** y el valor en cuestión.
Terminar todas las líneas del procedimiento almacenado con un **;** (salvo el begin).
declare nombre tipodato;
declare nombre tipodato default valor;

Para realizar operaciones con variables, deberemos indicar la palabra **set** e indicar la operación, terminando con un **;**
set edad = edad+10;

set edad = edad + edad;

En este caso, nuestro "return" será un **select** acompañado del nombre de la variable.

*Si el procedimiento almacenado presenta más de una línea, es necesario indicar donde empieza (begin) y donde acaba (end).
**Sebemos declarar antes de un procedimiento almacenado cual será el delimitador de bloque y este debemos ponerlo después del end. Tras el end, tenemos que volver a indicar que el limitar de bloque volverá a ser el ;

```
delimiter //

create procedure calcular_edad(agno_nacimiento int)
	begin
		declare agno_actual int default 2021;
		declare edad int;

		set edad = agno_actual - agno_nacimiento;
		select edad;
	end;//
delimiter ;
```

Título V: Triggers condicionales

Los triggers condicionales pueden ser útiles para controlar ciertos campos: para prevenir que usuarios introduzcan en el campo **edad** valores negativos, para prevenir que introduzcan en el campo **precio** un valor negativo o disparatado…

En este ejemplo, crearemos un trigger que controle registros ya creados del campo **precio** para que no se actualice en él ningún valor

disparatado y que si esto se intenta, no se cambie ningún valor y se conserve el antiguo:

```
delimiter //

create trigger revisa_precio_bu before update on productos for each row
	begin
		if(new.precio<0) then
			set new.precio=old.precio;
		elseif(new.precio>1000) then
			set new.precio=old.precio;
		end if;
	end;//

delimiter ;
```