INDICE

1. USO DEL WHERE, FROM, OR, AND, BETWEEN.

SELECT: Se usa para seleccionar las columnas/campos con las que se quiere operar.

FROM: Se usa posteriormente a SELECT para especificar de que relación/tabla se quieren seleccionar las columnas.

WHERE: Se usa para condicionar la consulta y reducirla, por ejemplo, si quisiésemos hacer una consulta a las columnas SECCIÓN, NOMBREARTÍCULO y PRECIO y quisiésemos que en solo muestre las filas que en la columna SECCIÓN tenga escrito DEPORTES sería:

```
SELECT SECCION, NOMBREARTÍCULO, PRECIO FROM PRODUCTOS WHERE SECCIÓN='DEPORTES';
```

Y nos mostraría las tres columnas de los registros donde SECCIÓN sea DEPORTES. Si quisiésemos que mostrase los registros donde SECCIÓN es DEPORTES, CERÁMICA y FERRETERÍA sería:

```
SELECT SECCIÓN, NOMBREARTÍCULO, PRECIO FROM PRODUCTOS WHERE SECCIÓN='DEPORTES' OR SECCIÓN='CERÁMICA' OR SECCIÓN='FERRETERÍA';
```

EJERCICIOS USO DEL WHERE, FROM, OR, AND, BETWEEN.

(Los ejercicios se hacen con la tabla PRODUCTOS y CLIENTES en PHPMyAdmin)

1. Realizar una consulta que muestre los campos "Empresa" y "Población" de la tabla "Clientes".

```
SELECT EMPRESA, POBLACIÓN FROM CLIENTES;
```

2. Realizar una consulta que muestre los artículos d la sección "Cerámica".

```
SELECT * FROM PRODUCTOS WHERE SECCIÓN="CERÁMICA";
```

3. Realizar una consulta que muestre los productos de la sección "Deportes" cuyo precio esté entre 100 y 200 €. En la consulta solo se mostrarán los campos "Nombre de artículo" y "Precio".

```
SELECT NOMBREARTÍCULO, PRECIO FROM PRODUCTOS WHERE SECCIÓN="DEPORTES" AND PRECIO BETWEEN 100 AND 300;
```

4. Realizar una consulta que muestre los productos cuyo país no sea España.

```
SELECT * FROM PRODUCTOS WHERE PAÍSDEORIGEN!="ESPAÑA";
```

5. Realizar una consulta que muestre los artículos españoles de la sección "Deportes" o aquellos cuyo precio sea superior a 350 € independientemente de cual sea su sección o país de origen.

```
SELECT * FROM PRODUCTOS WHERE SECCIÓN="DEPORTES" AND PAÍSDEORIGEN="ESPAÑA" OR PRECIO>350;
```

6. Realizar una consulta que muestre los productos cuya fecha esté entre 1/05/2001 y 15/12/2001. En la consulta solo se visualizarán los campos "Nombre de artículo", "Sección" y "Fecha".

```
SELECT NOMBREARTÍCULO, SECCIÓN, FECHA FROM PRODUCTOS WHERE FECHA BETWEEN '2001-05-01' AND '2001-12-15';
```

2. CONSULTAS CON CÁLCULOS:

Si queremos mostrar la columna NOMBREARTÍCULOS, la de PRECIO y otra que se llame IVA y contengo el 21% del precio:

```
SELECT NOMBREARTÍCULO, PRECIO, PRECIO*0.21 AS IVA FROM PRODUCTOS;
```

Si en vez de eso quisiésemos una tabla con el precio y el iva sumado sería:

```
PRECIO*1.21 AS PRECIO_CON_IVA.
```

ROUND(): Si ponemos ROUND(PRECIO*1.21,2) redondeará a dos decimales y no quedará tan largo el precio.

NOW(): Si incluimos en los campos NOW() AS DIA_DE_HOY, nos aparecerá otra columna que contiene la fecha y la hora actual.

DATEFIFF(): Nos devuelve la diferencia en días de dos fechas, en este caso sería DATEDIFF(NOW(),FECHA) y nos daría la columna.

DATE_FORMAT(): NOW() Nos devuelve la fecha completa y su hora, si queremos que solo muestre el día y el mes podemos hacerlo con DATE_FORMAT(NOW(),'%D-%M').

AVG(): Se usa para calcular la media de una columna numérica. Por ejemplo: AVG(PRECIO) AS MEDIA_PRECIO.

COUNT(): Cuenta el número de filas.

SUM(): Suma la columna.

MIN() y MAX(): Devuelve el valor mínimo o máximo.

ABS(): Valor absoluto.

SQRT(): Raíz cuadrada.

POWER(): Potencia.

RAND(): Número random.

3. GROUP BY Y ORDER BY.

GROUP BY:

Si queremos agrupar los artículos por SECCIÓN y que nos muestre las columnas SECCIÓN Y PRECIO y en PRECIO aparezca la suma de sus artículos:

```
SELECT SECCIÓN, SUM(PRECIOS) FROM PRODUCTOS GROUP BY SECCIÓN;
```

Si queremos lo mismo, pero en vez de agrupados por SECCIÓN, sea por PAÍSDEORIGEN:

```
SELECT PAÍSDEORIGEN, SUM(PRECIOS) FROM PRODUCTOS GROUP BY PAÍSDEORIGEN;
```

Si además queremos que se ordenen las sumas de los precios de menor a mayor, tendríamos que darle un alias a SUM(PRECIOS), en este caso SUMPRECIOS.

```
SELECT PAÍSDEORIGEN, SUM(PRECIOS) AS SUMPRECIOS FROM PRODUCTOS GROUP BY PAÍSDEORIGEN
ORDER BY SUMPRECIOS;
```

SELECT POBLACIÓN, COUNT(POBLACIÓN) AS CONTPOBLACIÓN FROM CLIENTES GROUP BY POBLACIÓN;

```
SELECT SECCIÓN, MAX(PRECIO) FROM PRODUCTOS GROUP BY SECCIÓN HAVING SECCIÓN="CONFECCIÓN";
```

ORDER BY:

Si quieres ordenarlos según la SECCIÓN, se pondría después de completar el WHERE,

```
ORDER BY SECCIÓN;
```

Si quieres ordenarlos por SECCIÓN y en cada sección se ordenen por precio pero en vez de menor a mayor, sea al revés,

```
ORDER BY SECCIÓN, PRECIO DESC;
```

EJERCICIOS CONSULTAS CON CÁLCULOS, GROUP BY Y ORDER BY.

1. Realizar una consulta sobre la tabla "Clientes" que muestre los campos "Dirección", "Teléfono" y "Población". Este último debe aparecer en la consulta con el nombre de "Residencia". Los registros aparecerán ordenados descendentemente por el campo "Población".

```
SELECT DIRECCIÓN, TELÉFONO, POBLACIÓN AS RESIDENCIA FROM CLIENTES ORDER BY RESIDENCIA
DESC;
```

2. Realizar una consulta que muestre que poblaciones hay en la tabla “Clientes”.

```
SELECT POBLACIÓN FROM CLIENTES;
```

3. Realizar una consulta de agrupación que muestre la media del precio de los artículos de todas las secciones. Mostrar en la consulta los campos sección y suma por sección.

```
SELECT SECCIÓN, AVG(PRECIO) AS MEDIA_PRECIO, SUM(PRECIO) AS SUMA_PRECIO FROM PRODUCTOS
GROUP BY SECCIÓN ORDER BY SUMA_PRECIO DESC;
```

4. Realizar una consulta de agrupación que muestre la media del precio de todas las secciones menos de juguetería. En la consulta deberán aparecer los campos “Sección” y “Media por sección”.

```
SELECT SECCIÓN, AVG(PRECIO) AS MEDIA_PRECIO FROM PRODUCTOS WHERE SECCIÓN!="JUGUETERÍA"
GROUP BY SECCIÓN ORDER BY MEDIA_PRECIO DESC;
```

5. Realizar una consulta que muestre cuantos artículos hay de la sección “Deportes”.

```
SELECT SECCIÓN, COUNT(SECCIÓN) AS ARTÍCULOS_DEPORTES FROM PRODUCTOS WHERE
SECCIÓN="DEPORTES";
```

6. Realizar una consulta que visualice los campos NOMBRE ARTÍCULO, SECCIÓN, PRECIO de la tabla PRODUCTOS y un campo nuevo que nombramos con el texto “DESCUENTO_7”. Debe mostrar el resultado de aplicar sobre el campo PRECIO un descuento de un 7 %. El formato del nuevo campo para debe aparecer con 2 lugares decimales.

```
SELECT NOMBREARTÍCULO, SECCIÓN, PRECIO, ROUND(PRECIO*0.93,2) AS PRECIO_CON_DTO7 FROM
PRODUCTOS;
```

7. Realizar una consulta visualizando los campos FECHA, SECCIÓN, NOMBRE ARTÍCULO y PRECIO de la tabla PRODUCTOS y un campo nuevo que nombramos con el texto “DTO2 €_EN_CERÁMICA”. Debe mostrar el resultado de aplicar sobre el campo PRECIO la resta de 2 € sólo a los artículos de la sección CERÁMICA. El formato del nuevo campo debe aparecer con 2 lugares decimales. Ordenar el resultado de la consulta por el campo FECHA descendente.

```
SELECT FECHA, NOMBREARTÍCULO, SECCIÓN, PRECIO, ROUND(PRECIO-2,2) AS DTO2_€_EN_CERÁMICA
FROM PRODUCTOS WHERE SECCIÓN='CERÁMICA' ORDER BY FECHA DESC;
```

8. Realizar una consulta visualizando los campos NOMBRE ARTÍCULO, SECCIÓN, PRECIO de la tabla PRODUCTOS y un campo nuevo que nombramos con el texto “PRECIO_AUMENTADO_EN_2”. Debe mostrar el PRECIO con un incremento de un 2% del PRECIO. Sólo debemos tener en cuenta los registros de la sección FERRETERÍA. El nuevo campo debe aparecer en Euros y con 2 lugares decimales.

```
SELECT NOMBREARTÍCULO, SECCIÓN, PRECIO, ROUND(PRECIO*1.02,2) AS PRECIO_AUMENTADO_EN_2
FROM PRODUCTOS WHERE SECCIÓN='FERRETERÍA';
```

4. UNION & UNION ALL.

A veces tendremos que hacer consultas de dos o más tablas, para unir las tablas usaremos UNION así:

```
SELECT * FROM TABLA1 WHERE SECCIÓN='DEPORTES' UNION SELECT * FROM TABLA1 WHERE
SECCION='CERÁMICA';
```

Si en vez de usar UNION usamos UNION ALL la consulta mostrará todos los datos aunque se repitan, es decir, el uso de UNION suprime los datos repetidos mostrándolos una única vez.

5. INNER JOIN, LEFT JOIN y RIGHT JOIN.

INNER JOIN: Esta sentencia fusiona dos tablas según su clave primaria y foránea respectivamente, y muestra los registros que coincidan, por ejemplo, en la tabla CLIENTES es clave primaria CÓDIGO CLIENTE y en la tabla PEDIDOS es clave foránea CÓDIGO CLIENTE, para que nos muestre los datos relacionados de ambas tablas escribimos:

```
SELECT * FROM CLIENTES INNER JOIN PEDIDOS ON CLIENTES.CÓDIGOCLIENTE=PEDIDOS.[CÓDIGO
CLIENTE] WHERE POBLACIÓN='MADRID';
```

Esta sentencia muestra los clientes QUE HAN HECHO PEDIDOS y todos sus datos, los clientes de la tabla CLIENTES que no hayan hecho pedidos no se mostrarán porque no están relacionados con la tabla PEDIDOS.

Si un nombre tiene espacios, en Access, se pone el nombre entre corchetes.

LEFT JOIN: Esta sentencia muestra los registros que coinciden, como ocurre con INNER JOIN, pero además muestra todos los registros de una de las tablas, en nuestro caso, la izquierda, CLIENTES:

```
SELECT * FROM CLIENTES LEFT JOIN PEDIDOS ON CLIENTES.CÓDIGOCLIENTE=PEDIDOS.[CÓDIGO
CLIENTE] WHERE POBLACIÓN='MADRID';
```

Ahora se mostrará lo mostrado con INNER JOIN y también los clientes que NO han hecho pedidos.

RIGHT JOIN: Esta sentencia mostraría sólo los registros que coincidan en ambas tablas y además las de CLIENTES, pero por la naturaleza de las tablas, no va a existir ningún pedido que no esté relacionado con un cliente de la tabla CLIENTES.

```
SELECT * FROM CLIENTES RIGHT JOIN PEDIDOS ON CLIENTES.CÓDIGOCLIENTE=PEDIDOS.[CÓDIGO
CLIENTE] WHERE POBLACIÓN='MADRID';
```

6. INSERT TO.

Creamos la tabla del ejemplo (SQLPlus o SQLDeveloper):

```
CREATE TABLE TABLA1 (
	DNI VARCHAR2(9) PRIMARY KEY,
	NOMBRE VARCHAR2(20),
	DIRECCION VARCHAR2(20),
```

```
TELEFONO NUMBER(9)
)
```

Para insertar datos en una tabla se usa la frase:

```
INSERT INTO NOMBRE_TABLA (NOMBRE_ATRIBUTO/CAMPO) VALUES (VALOR_A_INTRODUCIR);
```

Por ejemplo, si tenemos una TABLA1 con los campos DNI, NOMBRE, DIRECCION y TELEFONO sería:

```
INSERT INTO TABLA1 (DNI, NOMBRE, DIRECCION, TELEFONO) VALUES ('49338447B', 'RAUL',
'CALLE SALAMANCA', '676555384');
```

Si queremos meter varias filas en una sola sentencia sería:

```
INSERT INTO TABLA1 (DNI, NOMBRE, DIRECCION, TELEFONO) VALUES
('49338447B', 'RAUL', 'CALLE SALAMANCA', '676555384'),
('37449114M', 'JUAN', 'CALLE MANUEL', '633555283'),
('39114223B', 'ALBERTO', 'CALLE SALAMANCA', '633555053');
```

El tema de la creación de tablas continua en el punto "10. CREACIÓN DE TABLAS".

7. SUBCONSULTAS (ANY, ALL, IN, NOT IN).

Existen tres tipos de subconsulta:

- Escalonada.
- De lista.
- Correlacionada.

Escalonada:

```
SELECT NOMBREARTÍCULO, SECCIÓN, PRECIO FROM PRODUCTOS WHERE PRECIO>(SELECT AVG(PRECIO)
FROM PRODUCTOS);
```

Usamos la subconsulta hija como criterio, que es la que halla la media de los productos, y la ponemos entre paréntesis, y antes de ella hacemos la consulta padre que es para comparar el precio de los productos con la media.

De lista:

(ALL, ANY)

```
SELECT * FROM PRODUCTOS WHERE PRECIO> ALL (SELECT PRECIO FROM PRODUCTOS WHERE SECCIÓN='CERÁMICA');
```

En este caso en la consulta hija filtramos el precio de los productos de la sección CERÁMICA y en la consulta padre pedimos que seleccione todos los productos cuyo precio sea mayor que todos(ALL) los de la consulta hija. Es decir, los productos que tengan un precio mayor que el más caro de CERÁMICA.

Si usásemos ANY en vez de ALL, estaríamos pidiendo que nos mostrase todos los productos más caros que el más barato de CERÁMICA.

(IN, NOT IN)

En este ejemplo queremos mostrar los productos cuyas unidades superen las 20 vendidas en la tabla PRODUCTOSPEDIDOS, usamos IN.

```
SELECT NOMBREARTÍCULO, PRECIO FROM PRODUCTOS WHERE CÓDIGOARTÍCULO IN (SELECT CÓDIGOARTÍCULO FROM PRODUCTOSPEDIDOS WHERE UNIDADES>20);
```

Podríamos hacer la misma consulta usando INNER JOIN de esta manera:

```
SELECT NOMBREARTÍCULO, PRECIO FROM PRODUCTOS INNER JOIN PRODUCTOSPEDIDOS ON PRODUCTOS.CÓDIGOARTÍCULO=PRODUCTOSPEDIDOS.CÓDIGOARTÍCULO WHERE UNIDADES>20;
```

En este caso con INNER JOIN no haría falta que las tablas estuviesen relacionadas, con la subconsulta, sí. Aunque resulta más sencillo el código de la subconsulta.

En este ejemplo usamos el NOT IN para mostrar los clientes que han pagado con tarjeta o no han realizado pedidos, excluyendo los que han pagado al contado o aplazado.

```
SELECT EMPRESA, POBLACIÓN FROM CLIENTES WHERE CÓDIGOCLIENTE NOT IN (SELECT CÓDIGOCLIENTE FROM PEDIDOS WHERE FORMADEPAGO="CONTADO" OR FORMADEPAGO="APLAZADO");
```

Correlacionada:

Son subconsultas que hacen referencia a una columna de la consulta principal. Se pueden utilizar para realizar operaciones complejas que no se pueden realizar con una sola subconsulta.

8. CONSULTAS DE ACCION (UPDATE, CREATE, DELETE, INSERT INTO, SELECT INTO, DISTINCT, DISTINCTROW).

UPDATE:

Si queremos actualizar, por ejemplo, los precios de una sección de nuestra base de datos, en este caso sumar 10 a los precios de los artículos de la sección de deportes, se haría así usando la cláusula SET:

```
UPDATE PRODUCTOS SET PRECIO+=10 WHERE SECCIÓN=’DEPORTES’;
```

Si en vez de eso quisiésemos restarles a todos los productos 10 en su precio, sencillamente sería así:

```
UPDATE PRODUCTOS SET PRECIO-=10;
```

Si queremos cambiar la denominación de una sección sería:

```
UPDATE PRODUCTOS SET SECCIÓN=’DEPORTIVOS’ WHERE SECCIÓN=’DEPORTES’;
```

Creación de tablas (SELECT INTO, CREATE INTO):

Si quisiésemos crear una tabla sólo con los registros de clientes donde la población sea Madrid, se haría distinto en Access y en MySQL:

ACCESS

```
SELECT * INTO CLIENTES_MADRID FROM CLIENTES WHERE POBLACIÓN='MADRID';
```

MySQL

```
CREATE TABLE CLIENTES_MADRID SELECT * FROM CLIENTES WHERE POBLACIÓN=’MADRID’;
```

Eliminación (DELETE):

Si queremos eliminar los registros de clientes donde la población es Madrid:

```
DELETE FROM CLIENTES WHERE POBLACIÓN=’MADRID’;
```

Para complicarlo un poco más podríamos hacer la siguiente consulta:

```
DELETE FROM PRODUCTOS WHERE SECCIÓN=’DEPORTES’ AND PRECIO BETWEEN 20 AND 100;
```

Y eliminaría los productos de la sección de deportes cuyo precio se encuentre entre 20 y 100.

DISTINCT y DISTINCTROW:

```
SELECT EMPRESA FROM CLIENTES INNER JOIN ON
CLIENTES.CÓDIGOCLIENTE=PEDIDOS.CÓDIGOCLIENTE;
```

En la consulta anterior se nos mostraría los nombres de las empresas según los pedidos que han hecho, es decir si una empresa ha hecho cuatro pedidos, se mostrarían cuatro filas del mismo nombre de dicha empresa. Para evitar eso y que se nos muestre únicamente las empresas que han hecho pedidos una sola vez, se usaría la cláusula DISTINCT así:

```
SELECT DISTINCT EMPRESA FROM CLIENTES INNER JOIN ON
CLIENTES.CÓDIGOCLIENTE=PEDIDOS.CÓDIGOCLIENTE;
```

De esta forma pedimos que no se repitan los campos, sólo aparecerá el nombre de cada empresa una sola vez.

Con DISTINCTROW impedimos que al realizar una consulta muestre las filas que son idénticas, es decir, si por lo que sea hay dos productos con el mismo código de producto, mismo nombre, mismo precio, etc, con DISTINCTROW solo se mostrará una vez.

```
SELECT * FROM PRODUCTOS WHERE SECCIÓN=’FERRETERÍA’;
```

Así, en el caso de haber dos productos en la sección ferretería repetidos por error, se mostrarían ambos en la consulta, lo podemos evitar así:

```
SELECT DISTINCTROW * FROM PRODUCTOS WHERE SECCIÓN=’FERRETERÍA’;
```

Conclusión: con DISTINCT realizamos consultas en las que se pueden repetir los nombres o los datos y hacemos que sólo se muestre una vez cada uno. Con DISTINCTROW evitamos que en las consultas se muestren filas que se repiten completamente, sólo se muestran una vez.

Ejemplo:

Si quisiésemos eliminar los clientes que no han hecho pedidos de la base de datos, tendríamos que utilizar DISTINCTROW de la siguiente manera:

```
DELETE DISTINCTROW CLIENTES.*, PEDIDOS.CÓDIGOCLIENTE FROM CLIENTE LEFT JOIN PEDIDOS ON CLIENTES.CÓDIGOCLIENTE=PEDIDOS.CÓDIGOCLIENTE WHERE PEDIDOS.CÓDIGOCLIENTE IS NULL
```

REPASAR INNER JOIN, LEFT JOIN, ETC

INSERT INTO:

Para este ejemplo primero eliminamos todos los clientes donde la población sea Madrid:

```
DELETE CLIENTES FROM CLIENTES WHERE POBLACIÓN=’MADRID’;
```

Entonces queremos meter los clientes de la tabla clientes Madrid, donde sólo están los clientes de Madrid, en la tabla clientes, es decir, revertir lo anteriormente eliminado añadiendo la tabla CLIENTES MADRID a la tabla CLIENTES:

```
INSERT INTO CLIENTES SELECT * FROM CLIENTES_MADRID;
```

Si no queremos anexar todos los campos la consulta sería, por ejemplo:

```
INSERT INTO CLIENTES (CÓDIGOCLIENTE, EMPRESA, POBLACIÓN, TELÉFONO) SELECT CÓDIGOCLIENTE, EMPRESA, POBLACIÓN, TELÉFONO FROM CLIENTES_MADRID;
```

Especificamos en ambos casos los campos elegidos.

EJERCICIOS CONSULTAS DE ACCIÓN: UPDATE, CREATE, SELECT INTO.

1. Realizar una consulta de acción de creación de tabla a partir de la tabla CLIENTES, utilizando todos los campos de la tabla, pero únicamente los registros que sean de la población Madrid. El nuevo objeto lo nombramos con el texto “CLIENTES_DE_MADRID”. Ejecutamos la consulta.

```
CREATE TABLE CLIENTES_DE_MADRID SELECT * FROM CLIENTES HERE POBLACIÓN='MADRID';
```

2. Realizar una consulta de acción de creación de tabla a partir de la tabla PRODUCTOS, utilizando todos los campos de la tabla, pero sólo los registros que sean de la sección DEPORTES. El nuevo objeto – tabla lo nombramos con el texto “ARTÍCULOS_DE_DEPORTES”. Ejecutamos la consulta.

```
CREATE TABLE ARTÍCULOS_DE_DEPORTE SELECT * FROM PRODUCTOS WHERE SECCIÓN='DEPORTES';
```

3. Realizar una consulta de acción de creación de tabla a partir de la tabla PEDIDOS, utilizando todos los campos de la tabla, pero sólo los registros que tengan registrada la forma de pago TARJETA. El nuevo objeto – tabla lo nombramos con el texto “PEDIDOS_PAGADOS_CON_TARJETA”. Ejecutamos la consulta.

```
CREATE TABLE PEDIDOS_PAGADOS_CON_TARJETA SELECT * FROM PEDIDOS WHERE "FORMA DE PAGO"='TARJETA';
```

En Access los nombres con espacio se delimitan con corchetes [] en vez de comillas dobles "". En Access sería:

```
SELECT * INTO PEDIDOS_PAGADOS_CON_TARJETA FROM PEDIDOS WHERE [FORMA DE PAGO]='TARJETA';
```

4. Realizar una consulta que actualice los precios de la tabla ARTÍCULOS DE DEPORTE. La actualización consiste en calcular el IVA (21%) y mostrar en ese campo como resultado el precio con el IVA incluido. Ejecutar la consulta.

```
UPDATE PRODUCTOS SET PRECIO=PRECIO*1.21;
```

Con el código anterior actualizaríamos el precio en la propia tabla y en todas las posteriores consultas. Con el siguiente código crearíamos una consulta que nos muestra el precio con el IVA sumado, pero no lo modifica en la tabla:

```
SELECT PRECIO*1.21 AS PRECIO_CON_IVA FROM PRODUCTOS;
```

5. Realizar una consulta que actualice el campo DESCUENTO de la tabla PEDIDOS_PAGADOS_CON TARJETA. La actualización consiste poner a un 5% los descuentos que se muestran inferiores a esta cifra. Ejecutar la consulta.

```
UPDATE PEDIDOS_PAGADOS_CON_TARJETA SET DESCUENTO=0.05 WHERE DESCUENTO<0.05;
```

9. REFERENCIAS CRUZADAS.

Con las referencias cruzadas conseguimos mostrar consultas en tablas de forma distinta a la habitual. En este primer caso mostraremos en las columnas cada SECCIÓN, en las filas cada NOMBRE DE ARTÍCULO y en la llamada zona de totales, la suma de los precios. El código sería así:

```
TRANSFORM SUM(PRECIO) AS TOTAL

SELECT NOMBREARTÍCULO FROM PRODUCTOS GROUP BY NOMBREARTÍCULO

PIVOT SECCIÓN;
```

Tenemos que usar siempre la cláusula TRANSFORM para la zona de totales, la de GROUP BY para la zona de filas y la de PIVOT para la zona de columnas.

```
SELECT EMPRESA, POBLACIÓN, [FORMA DE PAGO] FROM CLIENTES INNER JOIN PEDIDOS ON
CLIENTES.CÓDIGOCLIENTE=PEDIDOS.[CÓDIGO CLIENTE];


TRANSFORM COUNT(POBLACIÓN) AS TOTAL

SELECT EMPRESA FROM PREVIA GROUP BY EMPRESA

PIVOT [FORMA DE PAGO];
```

EJERCICIOS DE REFERENCIAS CRUZADAS.

1. Realiza una consulta de referencias cruzadas que muestre cuántos artículos hay en la tabla de productos por cada año. El resultado obtenido sea el siguiente: (*Ayuda: Para agrupar por año utiliza la función Format: Format(FECHA,"yyyy")*)

PAÍSDEORIGEN	TOTAL	2000	2001	2002
CHINA	5	3	1	1
ESPAÑA	11	2	3	6
FRANCIA	2		1	1
ITALIA	4	1	1	2
JAPÓN	5	2	3	
MARRUECOS	3	1	1	1
SUECIA	1		1	
TAIWÁN	1			1
TURQUÍA	1		1	
USA	7	4	1	2

```
TRANSFORM COUNT(PRECIO) AS TOTAL

SELECT PAÍSDEORIGEN FROM PRODUCTOS GROUP BY PAÍSDEORIGEN

PIVOT FORMAT(FECHA, "yyyy");
```

2. Realiza una consulta de referencias cruzadas que muestre cuántos artículos importados y no importados han sido enviados, y cuántos importados y no importados hay sin enviar. El resultado sea el siguiente:

IMPORTADO	-1	0
☑	5	12
☐		1

```
SELECT PRODUCTOS.CÓDIGOARTÍCULO, IMPORTADO, ENVIADO FROM PRODUCTOS INNER JOIN
(PEDIDOS INNER JOIN PRODUCTOSPEDIDOS ON PEDIDOS.[NÚMERO DE
PEDIDO]=PRODUCTOSPEDIDOS.[NÚMERO DE PEDIDO]) ON
PRODUCTOS.CÓDIGOARTÍCULO=PRODUCTOSPEDIDOS.[CÓDIGO ARTÍCULO];
```

Hacemos la consulta PREVIA y después:

```
TRANSFORM COUNT(CÓDIGOARTÍCULO) AS TOTAL

SELECT IMPORTADO FROM PREVIA GROUP BY IMPORTADO

PIVOT ENVIADO;
```

10. CREACIÓN DE TABLAS.

Esto es una continuación del punto "6. INSERT TO" donde se ve brevemente la creación de tablas y la inserción de datos.

Para crear una tabla en MySQL con los campos "id alumno", "nombre", "apellido", "edad", "fecha de nacimiento" y "carnet" y asignar los tipos de valor correctos se haría de la siguiente forma:

```
CREATE TABLE TABLA_1 (
        ID_ALUMNO INT AUTO_INCREMENT PRIMARY KEY,
        NOMBRE VARCHAR(20),
        APELLIDO VARCHAR(20),
        EDAD INT,
        FECHA_DE_NACIMIENTO DATE,
        CARNET BOOLEAN
        );
```

En este caso "id alumno" es un número entero que se va auto incrementando con cada registro que se incluya. A su vez, "id alumno" es clave primaria de la tabla.

11. MODIFICAR TABLAS Y CAMPOS.

Si queremos modificar una tabla, por ejemplo, añadiendo un campo, utilizamos la cláusula ALTER:

```
ALTER TABLE TABLA_1 ADD COLUMN DNI VARCHAR(9);
```

Si queremos modificar algún campo, lo haríamos así:

```
ALTER TABLE TABLA_1 ALTER COLUMN DNI INT(8);
```

En el caso anterior el DNI iría sin la letra final.

Si queremos que, al no introducir ningún DNI, no aparezca en el registro la cláusula NULL, tendríamos que poner, después de indicar el tipo de dato, la cláusula NOT NULL o indicar un default poniendo, por ejemplo, SET DEFAULT 'DESCONOCIDO', en caso de que DNI fuese VARCHAR, así:

```
ALTER TABLE TABLA_1 ALTER COLUMN DNI SET DEFAULT 'DESCONOCIDO';
```

Para eliminar el valor por defecto, usaríamos DROP DEFAULT.

12. INDICES.

INDICES DE CLAVE PRIMARIA:

Al crear una clave primaria estamos creando un índice, las claves primarias no pueden ser iguales entre un registro y otro y no pueden ser NULL, podemos nombrar una clave primaria de dos formas, al crear el campo en cuestión:

```
CREATE TABLE TABLA_1 (DNI VARCHAR(9) PRIMARY KEY);
```

O modificando con la clausula ALTER un campo ya creado:

```
ALTER TABLE TABLA_1 ADD PRIMARY KEY (DNI);
```

También podemos nombrar dos campos como PK:

```
ALTER TABLE TABLA_1 ADD PRIMARY KEY (NOMBRE, APELLIDO);
```

INDICES ORDINARIOS:

Esta forma de índice si permite duplicados y ser NULL. Para crear índices sobre campos que no son PK usamos la palabra reservada INDEX:

```
CREATE INDEX MI_INDICE ON TABLA_1 (APELLIDO);
```

INDICES UNICOS:

Los índices únicos son exactamente iguales a los ordinarios con al excepción de que no permiten duplicados, es decir, un campo no puede tener dos registros iguales al ser índice único. Se usaría la cláusula UNIQUE:

```
CREATE UNIQUE INDEX MI_INDICE ON TABLA_1 (APELLIDO);
```

INDICES COMPUESTOS:

Este índice permite estar formado por dos o más campos, pero no permite que todos sus campos estén duplicados a la vez en otro registro, es decir, si usamos NOMBRE y APELLIDO, dos registros podrán tener mismo nombre o mismo apellido, pero no ambos iguales. Se declararía igual que el índice anterior, pero añadiendo los campos en cuestión:

```
CREATE UNIQUE INDEX MI_INDICE ON TABLA_1 (NOMBRE, APELLIDO);
```

ELIMINACIÓN DE INDICES:

Para eliminar los índices que no usan PK usamos la cláusula DROP y un código casi idéntico al de su creación, pero sin nombrar los campos que usa el índice:

```
DROP INDEX MI_INDICE ON TABLA_1;
```

Para eliminar un índice con clave primaria se debe saber el nombre que se le ha dado a dicho índice, en nuestro caso si usamos DNI como PK en la tabla TABLA_1 Access nombra nuestro índice así: "Index_D0F75744_461F_472E". Entonces, para eliminarlo escribiríamos lo siguiente, usando la palabra reservada CONSTRAINT (no válida para MySQL):

```
ALTER TABLE TABLA_1 DROP CONSTRAINT Index_D0F75744_461F_472E;
```

En MySQL sería mucho más sencillo:

```
ALTER TABLE TABLA_1 DROP PRIMARY KEY;
```

13. TRIGGERS.

Un "trigger" o disparador genera un evento antes o después de insertar, actualizar o eliminar información en una tabla. Por ejemplo, si queremos crear un trigger que se ejecute después (after) de insertar en la tabla productos un registro, sería:

```
CREATE TRIGGER PRODUCTOS_AI AFTER INSERT ON PRODUCTOS…
```

EJEMPLO TRIGGER REGISTROS (INSERT):

Entonces, para insertar los datos del nuevo registro en otra tabla llamada REGISTROS_PRODUCTOS primero creamos la tabla:

```
CREATE TABLE REGISTROS_PRODUCTOS (
        CÓDIGOARTÍCULO VARCHAR(25),
        NOMBREARTÍCULO VARCHAR(30),
        PRECIO INT(4),
        INSERTADO DATETIME
);
```

Y ahora creamos el trigger que haga que al insertar un nuevo registro en la tabla PRODUCTOS, se inserte otro registro automáticamente en la tabla REGISTROS_PRODUCTOS con los respectivos datos:

```
CREATE TRIGGER PRODUCTOS_AI AFTER INSERT ON PRODUCTOS FOR EACH ROW INSERT INTO
REG_PRODUCTOS
	(CÓDIGOARTÍCULO, NOMBREARTÍCULO, PRECIO, INSERTADO)
	VALUES
	(NEW.CÓDIGOARTÍCULO, NEW.NOMBREARTÍCULO, NEW.PRECIO, NOW());
```

Con esto ya podríamos insertar un nuevo registro en la tabla PRODUCTOS y que se registrasen los datos automáticamente en la tabla REGISTROS_PRODUCTOS.

EJEMPLO TRIGGER CAMBIOS (UPDATE):

Creamos la tabla donde se almacenarán los datos antiguos antes de la actualización y los datos nuevos una vez actualizados los registros:

```
CREATE TABLE PRODUCTOS_ACTUALIZADOS (
	ANTERIOR_CÓDIGOARTÍCULO VARCHAR(4), NUEVO_CÓDIGOARTÍCULO VARCHAR(4),
	ANTERIOR_NOMBREARTÍCULO VARCHAR(25), NUEVO_NOMBREARTÍCULO VARCHAR(25),
	ANTERIOR_SECCIÓN VARCHAR(15), NUEVO_SECCIÓN VARCHAR(25),
	ANTERIOR_PRECIO INT(4), NUEVO_PRECIO INT(4),
	ANTERIOR_IMPORTADO VARCHAR(15), NUEVO_IMPORTADO VARCHAR(15),
	ANTERIOR_PAÍSDEORIGEN VARCHAR(15), NUEVO_PAÍSDEORIGEN VARCHAR(15),
	ANTERIOR_FECHA DATE, NUEVO_FECHA DATE,
	USUARIO VARCHAR(15), FECHA_MODIFICACION DATE
	);
```

Una vez creada la tabla, procedemos con la creación del trigger:

```
CREATE TRIGGER ACTUALIZA_PRODUCTOS_BU BEFORE UPDATE ON PRODUCTOS FOR EACH ROW INSERT
INTO PRODUCTOS_ACTUALIZADOS (
	ANTERIOR_CÓDIGOARTÍCULO, NUEVO_CÓDIGOARTÍCULO,
	ANTERIOR_NOMBREARTÍCULO, NUEVO_NOMBREARTÍCULO,
	ANTERIOR_SECCIÓN, NUEVO_SECCIÓN,
	ANTERIOR_PRECIO, NUEVO_PRECIO,
	ANTERIOR_IMPORTADO, NUEVO_IMPORTADO,
	ANTERIOR_PAÍSDEORIGEN, NUEVO_PAÍSDEORIGEN,
	ANTERIOR_FECHA, NUEVO_FECHA,
```

```
        USUARIO, FECHA_MODIFICACION
        )
        VALUES (
        OLD.CÓDIGOARTÍCULO, NEW.CÓDIGOARTÍCULO,
        OLD.NOMBREARTÍCULO, NEW.NOMBREARTÍCULO,
        OLD.SECCIÓN, NEW.SECCIÓN,
        OLD.PRECIO, NEW.PRECIO,
        OLD.IMPORTADO, NEW.IMPORTADO,
        OLD.PAÍSDEORIGEN, NEW.PAÍSDEORIGEN,
        OLD.FECHA, NEW.FECHA,
        CURRENT_USER, NOW()
        );
```

Con la tabla para almacenar las actualizaciones y el trigger creado, creamos dos actualizaciones de ejemplo:

```
UPDATE PRODUCTOS SET PRECIO=PRECIO+500 WHERE CÓDIGOARTÍCULO='AR11';
UPDATE PRODUCTOS SET PAÍSDEORIGEN='FINLANDIA' WHERE CÓDIGOARTÍCULO='AR11';
```

Con estas dos sentencias se almacenarán dos registros en la tabla PRODUCTOS_ACTUALIZADOS, en el primer registro sólo cambiará PRECIO y en el segundo registro sólo cambiará PAÍSDEORIGEN.

EJEMPLO TRIGGER ELIMINACIÓN (DELETE):

Creamos la tabla donde se van a almacenar los registros eliminados:

```
CREATE TABLE PROD_ELIMINADOS (
        CÓDIGOARTÍCULO VARCHAR(5),
        NOMBREARTÍCULO VARCHAR(15),
        PAÍSDEORIGEN VARCHAR(15),
        PRECIO INTEGER,
        SECCIÓN VARCHAR(15)
        );
```

Entonces creamos el trigger de eliminación:

```
CREATE TRIGGER ELIM_PROD_AD AFTER DELETE ON PRODUCTOS FOR EACH ROW INSERT INTO
PROD_ELIMINADOS (
	CÓDIGOARTÍCULO,
	NOMBREARTÍCULO,
	PAÍSDEORIGEN,
	PRECIO,
	SECCIÓN)
	VALUES
	(OLD.CÓDIGOARTÍCULO,
	OLD.NOMBREARTÍCULO,
	OLD.PAÍSDEORIGEN,
	OLD.PRECIO,
	OLD.SECCIÓN);
```

Una vez creado el trigger y la tabla para almacenar los eventos del trigger, eliminamos un registro como ejemplo:

```
DELETE FROM PRODUCTOS WHERE CÓDIGOARTÍCULO='AR26';
```

Y deberían almacenarse sólo los valores que hemos establecido en la tabla PROD_ELIMINADOS.

ELIMINACIÓN Y MODIFICACIÓN DE TRIGGERS:

Si queremos modificar el trigger anterior para que incluya el usuario que ha realizado la eliminación y la fecha de eliminación, primero modificamos la tabla del trigger añadiendo los campos USUARIO y FECHA_ELIM:

```
ALTER TABLE PROD_ELIMINADOS ADD COLUMN (USUARIO VARCHAR(15), FECHA_ELIM DATE);
```

Una vez modificada la tabla, modificamos el trigger, como no existe la sentencia ALTER TRIGGER, lo que tendríamos que hacer es eliminarlo y volverlo a crear, pero se puede escribir todo en un mismo código:

```
DROP TRIGGER IF EXISTS ELIM_PROD_AD;
CREATE TRIGGER ELIM_PROD_AD AFTER DELETE ON PRODUCTOS FOR EACH ROW INSERT INTO
PROD_ELIMINADOS (
	CÓDIGOARTÍCULO,
	NOMBREARTÍCULO,
```

```
PAÍSDEORIGEN,
PRECIO,
SECCIÓN,
USUARIO,
FECHA_ELIM)
VALUES
(OLD.CÓDIGOARTÍCULO,
OLD.NOMBREARTÍCULO,
OLD.PAÍSDEORIGEN,
OLD.PRECIO,
OLD.SECCIÓN,
CURRENT_USER,
NOW());
```

Después realizamos alguna eliminación en la tabla PRODUCTOS como estas dos:

```
DELETE FROM PRODUCTOS WHERE CÓDIGOARTÍCULO='AR25';
DELETE FROM PRODUCTOS WHERE CÓDIGOARTÍCULO='AR39';
```

Y veremos como en la tabla PROD_ELEIMINADOS aparecen dos nuevos registros, esta vez con el usuario que realizó la eliminación y la fecha.

14. PROCEDIMIENTOS ALMACENADOS.

Los procedimientos almacenados son una manera de automatizar sentencias y comprimirlas para a la hora de querer hacer algo, sólo tener que introducir ciertos valores pasándolos por parámetros o simplemente darle a un botón.

Por ejemplo, si queremos cambiar el precio de un producto de la tabla PRODUCTOS haríamos el siguiente procedimiento:

```
CREATE PROCEDURE ACTUALIZA_PRODUCTOS (N_PRECIO DOUBLE, CODIGO VARCHAR(4))
UPDATE PRODUCTOS SET PRECIO=N_PRECIO WHERE CÓDIGOARTÍCULO=CODIGO;
```

En la primera sentencia creamos el procedimiento con las palabras reservadas CREATE PROCEDURE y estableciendo los parámetros a introducir, en la segunda sentencia le damos una función a los parámetros. Entonces, para llamar al procedimiento y establecer un nuevo precio

```
CALL ACTUALIZA_PRODUCTOS(37, 'AR01');
```

Entonces el precio del producto con código AR01 será 37, si queremos volver al mismo precio, tendríamos que hacer lo mismo introduciendo el precio anterior:

```
CALL ACTUALIZA_PRODUCTOS(6.6280, 'AR01');
```

Si queremos eliminar el procedimiento escribiríamos:

```
DROP PROCEDURE ACTUALIZA_PRODUCTOS;
```

Si queremos crear un procedimiento almacenado más complejo, por ejemplo, que nos devuelva la edad que tenemos según nuestro año de nacimiento, lo haríamos de la siguiente forma declarando variables y estableciendo un bloque de código con BEGIN y END:

```
CREATE PROCEDURE CALCULA_EDAD (AGNO_NACIMIENTO INT)
BEGIN
	DECLARE AGNO_ACTUAL INT DEFAULT 2024;
	DECLARE EDAD INT;
	SET EDAD=AGNO_ACTUAL-AGNO_NACIMIENTO;
	SELECT EDAD;
END;
```

En este caso tenemos que poner obligatoriamente el ";" al final del bloque de código, después del END, o en su caso, podemos usar un delimitador con la palabra reservada DELIMITER y estableciendo el símbolo delimitador, que por convención suelen ser "//" o "$$". Sería así:

```
DELIMITER $$
CREATE PROCEDURE CALCULA_EDAD (AGNO_NACIMIENTO INT)
BEGIN
	DECLARE AGNO_ACTUAL INT DEFAULT 2024;
	DECLARE EDAD INT;
	SET EDAD=AGNO_ACTUAL-AGNO_NACIMIENTO;
	SELECT EDAD;
END;$$
DELIMITER;
```

Y cerrar el DELIMITER al final del código.

Creamos un trigger que compruebe si el nuevo precio establecido es menor que 0 o mayor que 1000, en cuyo caso, establezca el valor en 0 o en 1000;

```
DELIMITER $$
CREATE TRIGGER COMPRUEBA_PRECIO_BU BEFORE UPDATE ON PRODUCTOS FOR EACH ROW
BEGIN
	IF (NEW.PRECIO<0) THEN
		SET NEW.PRECIO=0;
	ELSEIF (NEW.PRECIO>1000) THEN
		SET NEW.PRECIO=1000;
	END IF;
END;$$
DELIMITER;
```

Entonces, una vez creado este trigger, ponemos en consola, por ejemplo:

```
UPDATE PRODUCTOS SET PRECIO=1800 WHERE CÓDIGOARTÍCULO=’AR03’;
```

Entonces, el trigger establecerá el precio en 1000.

15. VISTAS.

Las vistas nos permiten realizar una especie de consultas predefinidas sin tener que utilizar el motor de búsqueda de la base de datos.

Por ejemplo, si queremos crear una vista que nos muestre las columnas NOMBREARTÍCULO, SECCIÓN y PRECIO de los productos de la sección DEPORTES, sería asó:

```
CREATE VIEW ART_DEPORTES AS
SELECT NOMBREARTÍCULO, SECCIÓN, PRECIO FROM PRODUCTOS WHERE SECCIÓN='DEPORTES';
```

Con esto ya tendríamos la vista creada y podemos consultarla, las vistas son fieles a los datos reflejados en la tabla original, es decir, si un valor cambia en la tabla, también se actualiza en la vista.

Las vistas permiten modificaciones con la palabra reservada ALTER. Por ejemplo, si queremos que también muestre la columna PAÍSDEORIGEN:

```
ALTER VIEW ART_DEPORTES AS

SELECT NOMBREARTÍCULO, SECCIÓN, PRECIO, PAÍSDEORIGEN FROM PRODUCTOS WHERE
SECCIÓN='DEPORTES';
```

Para eliminar la vista usamos una vez más la palabra reservada DROP:

```
DROP VIEW ART_DEPORTES;
```