**COMANDO DE SQL:**

DDL: Data definition Lenguages

\* Crear tablas

\* Elminar tablas

\* Modificar tablas

DML: Data manipulation Lenguege

\* Seleccionar registros de una BBDD

\* Consulta de selección y acción

DCL: Data Control Lenguages

\* Proporcionar seguridad a la información de la BBDD

TCL: Transation Control Lenguages

\*Gestión de los cambios de los datos

**COMANDOS DEL DDL:**

\* CREATE

\* ALTER

\* DROP

\* TRUNCATE

**COMANDOS DML:**

\* SELECT

\* INSERT

\* UPDATE

\* DELETE

**COMANDOS DCL:**

\* GRANT

\* REVOKE

**COMANDOS TCL:**

\* COMMIT

\* ROLLBACK

\* SAVEPOINT

**CLÁUSULAS DE SQL:**

\* 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.

\* ORDEN BY: Ordena los registros seleccionados en función de un campo.

\*\*\*En el caso de que una consulta requiera todas las cláusulas entonces su orden sería:

\* comando + from + where + group by + having + orden by

**INSTRUCCIÓN SQL:**

Es la unión de:

COMANDO + CLÁUSULAS + OPERADORES + FUNCIONES

**###### IMPORTAR LAS TABLAS A MYSQL ########**

UNA VEZ ESTAMOS EN ACCESS LE DAMOS A CLIC A LA TABLA QUE QUEREMOS EXPORTAR Y LE DAMOS A EXCEL. (OTRA FORMA ES IR A DATOS EXTERNOS Y CLIC EN EXCEL)

DESPUÉS, ABRIMOS EL ARCHIVO DE EXCEL Y LO GUARDAMOS EN FORMATO .ods (openDocument) PARA PODER IMPORTARLO EN MYSQL

**### CREAR INSTRUCCIÓN SQL EN ACCESS ###**

PARA CREAR UNA CONSULTA SQL EN ACCESS LE DAMOS A CREAR/DISEÑO DE CONSULTA/VER Y SELECCIONAMOS SQL

DESPUÉS SOLAMENTE LE DAMOS EN EJECUTAR Y YA NOS MOSTRARÁ TODA LA INFORMACIÓN DE NUESTRA INSTRUCCIÓN

**### CREAR INSTRUCCIÓN SQL EN MYSQL (PHPMYADMIN) ###**

SELECCIONAR NUESTRA BBBDD Y CLIC EN SQL, ESCRIBIR LA INSTRUCCIÓN Y DARLE A CONTINUAR.

**### OPERADORES DE COMPARACIÓN ###**

<, >, =, >=, <=,

<> distinto que

BETWEEN, entre, utlizado para especificar rangos de valores.

LIKE, como, utilizado con caracteres comodín. (se puede utilizar el LIKE “letra%” para saber si algo empieza por una letra)

In, en, utlizado para especificar registros en un campo en concreto.

Any: ALGUNO

All: TODOS

**### Operadores lógicos ###**

AND, OR, NOT

**###### COMANTARIO IMPORTANTE AL TENER NOMBRES DE COLUMNAS CON ESPACIOS #####**

Al momento de hacer consultas los nombres de las columnas no deben tener espacios, si es el caso entonces escribir lo siguiente `nombre columna`

\*\*\*\*\* EN ACCESS SE ENCIERRAN CON [ ]

\*\*\*\*\* las comillas `` salen presionando Alt Gr + }

**###### COMENTARIO IMPORTANTE ACERCA DE LOS FORMATOS DE LOS CAMPOS ######**

LOS CAMPOS DEBEN TENER EL FORMATO CORRECTO PORQUE SINO MYSQL PUEDE HACER CONSULTAS ERRONEAS.

POR EJEMPLO: EL CAMPO PRECIO NO DEBERÍA IR EN FORMATO VARCHAR (TEXTO) PORQUE AL MOMENTO DE HACER CONSULTAS HABRÁN FALLOS AL REALIZAR UNA CONSULTA PORQUE NO LO DETECTA COMO VALOR NÚMERO

\*\* CORREGIR EL FORMATO (EN ESTE CASO DEL CURSO) EN EL EXCEL Y LUEGO VOLVER A IMPORTAR LA TABLA

**\*\*\*\* LOS CRITERIOS DE TIPO STRING DEBEN IR EN COMILLAS ####**

**####### ORDER BY ########**

La cláusula ORDER BY permite ordenar varias categorías, por ejemplo, puedo ordenar primero por el tipo de grupo que pertenece un producto y luego ese grupo lo puedo ordenar por su precio

ORDEN BY x,y asc/desc

**####### CONSULTAS DE AGRUPACIÓN O TOTALES #######**

FUNCIONES DE AGREGADO:

\* AVG: Calcula el promedio de un campo

\* COUNT: Cuenta los registros de un campo

\* SUM: Suma los valores de un campo

\* MAX: Devuelve el máximo de un campo

\* MIN: Devuelve el mínimo de un campo

LAS FUNCIONES DE AGREGADO SE UTILIZAN DESPUÉS DEL SELECT Y ANTES DEL CAMPO EN LA QUE VAMOS A APLICAR LA FUNCION POR EJ: SELECT SECCIÓN, SUM(PRECIO) ...

\*\*\*la cláusula WHERE no debe contener funciones de agrupación o agregación, mientras que la cláusula HAVING siempre contiene funciones de agregación.

**------------ EJERCICIO SUM ---------------**

\*\*\*AQUI VAMOS A AGRUPAR POR SECCIÓN Y SUMAREMOS TODOS LOS PRECIOS DE CADA SECCIÓN

SELECT SECCIÓN, SUM(PRECIO) FROM productos GROUP BY SECCIÓN

\*\*\*\*AL AGRUPAR POR SECCIÓN Y COLOCAMOS UN TERCER CAMPO (NOMBRE ARTICULO) EN MYSQL APARECERÁ EL PRIMER ARTICULO DE CADA SECCIÓN

\*\*\*\*EN ACCESS, SIMPLEMENTE NO APARECERÁN LAS CONSULTAS O DIFICILES DE INTERPRETAR

**####### EL ALIAS NOS PERMITE CAMBIARLE EL NOMBRE A UN CAMPO ########**

**------------ EJERCICIO SUM Y AS ------------**

La cláusula AS nos permite cambiarle el nombre a un campo

\*AQUÍ VAMOS A ORDENAR POR LA SUMA DE LOS PRECIOS DE LOS ARTICULOS, PERO COMO NO EXISTE ESA COLUMNA ENTONCES LE COLOCAMOS UN ALIAS PARA PODER UTLIZAR EL RESULTADO DE SUM(PRECIO)

SELECT SECCIÓN, SUM(PRECIO) AS SUMA\_PRODUCTOS FROM productos GROUP BY SECCIÓN ORDER BY SUMA\_PRODUCTOS ASC

**------------ EJERCICIO CON HAVING Y AVG ------------**

AGRUPAR POR SECCIÓN LOS PRODUCTOS DE CERAMICA Y DEPORTES Y CALCULAR EL PROMEDIO DE SUS PRECIOS

AQUÍ COMO ESTAMOS TRABAJANDO CON

SELECT SECCIÓN, AVG(PRECIO) AS PROMEDIO\_PRODUCTOS FROM productos GROUP BY SECCIÓN HAVING SECCIÓN = "CERÁMICA" OR SECCIÓN = "DEPORTES" ORDER BY PROMEDIO\_PRODUCTOS ASC

**------------ EJERCICIO CON COUNT Y AS ------------**

CONTAR LA CANTIDAD DE CLIENTES DE UNA CIUDAD

SELECT POBLACIÓN, COUNT(`CÓDIGO CLIENTE`) AS CANTIDAD\_CLIENTES FROM clientes GROUP BY POBLACIÓN

**------------ EJERCICIO MAX Y MIN ------------**

EL PRECIO MÁS CARO DEL ARTÍCULO DE LA SECCIÓN CONFECCIÓN

SELECT SECCIÓN, MAX(PRECIO) AS PRECIO\_MAX FROM productos GROUP BY SECCIÓN HAVING SECCIÓN = "CONFECCIÓN"

SELECT SECCIÓN, MIN(PRECIO) AS PRECIO\_MIN FROM productos GROUP BY SECCIÓN HAVING SECCIÓN = "CONFECCIÓN"

**######## CONSULTAS DE CÁLCULOS (SE APLICAN EN REGISTROS INDIVIDUALES Y NO EN GRUPOS) O CONSULTAS CON CAMPO CALCULADO**

CADA GESTOR DE BBDD TIENE SUS PROPIAS CONSULTAS DE CÁLCULOS (ALGUNA QUE OTRA COMÚN)

Now(): día y hora actuales

Datediff(): diferencia entre dos fechas, recibe tres argumentos, el primero es el tipo de comparación si por día “D”, mes “M”, etc y los otros dos son las fechas a comparar. EN MYSQL SOLO RECIBE DOS PARAMETROS.

Date\_format(): formatear los resultados (‘%D-%M-%Y’)

Concat(): Concatenar (se suele usar para concatenar textos)

Round(): redondear resultados, le podemos pasar un segundo parámetro con la cantidad de decimales a mostrar

Truncate():

**------- EJERCICIO CON ROUND Y SOLO CÁLCULOS ------**

SELECT `NOMBRE ARTÍCULO`, PRECIO, ROUND(PRECIO\*1.21,2) AS PRECIO\_IVA FROM productos

SELECT `NOMBRE ARTÍCULO`,`PRECIO`, PRECIO-3 AS PRECIO\_DESCUENTO FROM productos

**-------- EJERCICIO CON DATE Y DEMÁS-------**

SELECT `NOMBRE ARTÍCULO`, PRECIO, FECHA, Date\_Format(Now(), "‘%D-%M-%Y’") AS FECHA\_ACT, DateDiff(Now(),FECHA) AS DIAS\_DIFERENCIA FROM productos WHERE SECCIÓN = "DEPORTES"

**########### CONSULTAS MULTITABLA / CONSULTAS DE UNIÓN #############**

UNIÓN EXTERNA:

**LAS TABLAS DEBEN TENER EL MISMO TAMAÑO Y TIPOS DE DATOS COMPATIBLES.**

* UNION: (NO MUESTRA TODOS LOS REGISTROS REPETIDOS)
* UNION ALL: (MUESTRA TODOS LOS REGISTROS REPETIDOS)
* EXCEPT:
* INTERSECT:
* MINUS: A-B MUESTRA LOS REGISTROS QUE ESTÁN PRESENTES EN A Y NO EN B

\*\*\* LOS TRES ÚLTIMOS NO SON SOPORTADOS POR ALGUNAS BBDD EN MYSQL Y ACCESS NO LOS ACEPTAN

UNIÓN INTERNA:

**LOS JOIN PERMITEN HACER CONSULTAS ENTRE TABLAS RELACIONADAS**

**EL ON DEBE IR PARA ESPECIFICAR LAS COLUMNAS EN COMÚN**

* INNER JOIN: REFLEJA LA INFORMACIÓN EN COMÚN ENTRE DOS TABLAS
* LEFT JOIN: NOS DEVUELVE LA INFORMACIÓN COMÚN ENTRE DOS TABLAS Y LA INFORMACIÓN DE LA PRIMERA TABLA
* RIGHT JOIN: INFORMACIÓN COMÚN E INFORMACIÓN DE LA SEGUNDA TABLA

---------- EJERCICIOS UNION Y UNION ALL ------------

SELECT \* FROM productos WHERE SECCIÓN = "DEPORTES" UNION SELECT \* FROM productosnuevos WHERE SECCIÓN="DEPORTES DE RIESGO"

SELECT \* FROM productos WHERE SECCIÓN = "CONFECCIÓN" UNION ALL SELECT \* FROM productosnuevos WHERE SECCIÓN = "DEPORTES"

\*\*SI SOLAMENTE SE QUIEREN TRABAJAR CON X CANTIDAD DE CAMPOS ENTONCES LA OTRA CONSULTA DEBE TRABAJAR CON LA MISMA CANTIDAD, POR EJEMPLO:

SELECT PRECIO,NOMBRE FROM productos UNION SELECT PRECIO,NOMBRE FROM productosNuevos…

**----------- EJERCICIOS JOIN EN ACCESS ------------**

SELECT \* FROM CLIENTES INNER JOIN PEDIDOS ON CLIENTES.`CÓDIGO CLIENTE`= PEDIDOS.`CÓDIGO CLIENTE` WHERE POBLACIÓN = "MADRID" ORDER BY [NÚMERO DE PEDIDO]

\*\* TOTAL DE PEDIDOS QUE HIZO UN CLIENTE

SELECT CLIENTES.[CÓDIGO CLIENTE], COUNT([NÚMERO DE PEDIDO]) FROM CLIENTES INNER JOIN PEDIDOS ON CLIENTES.[CÓDIGO CLIENTE] = PEDIDOS.[CÓDIGO CLIENTE] GROUP BY CLIENTES.[CÓDIGO CLIENTE]

\*\* REALIZADA EN ACCESS, COMO AMBAS TABLAS TIENEN EL MISMO NOMBRE DE COLUMNA ENTONCES SE ESPECÍFICA CON SU RESPECTIVA TABLA.

\*\* EN MYSQL HAY QUE RELACIONAR LAS TABLAS PORQUE AL IMPORTARLAS NO QUIERE DECIR QUE YA SE ESTÉN RELACIONANDO

**############ CREAR RELACIÓN ENTRE TABLAS EN MYSQL ############**

* HAY QUE ESPECIFICAR LA COLUMNA POR LA CUAL LAS TABLAS SE VAN A RELACIONAR Y ESPECIFICAR CUALES SON SUS LLAVES PRIMARIAS
* PARA LAS LLAVES PRIMARIAS:
  + LE DAMOS A LA TABLA, LUEGO A ESTRUCTURA, INDICES…
  + TIPO INDICE = PRIMARY Y ESPECIFICAR LA COLUMNA Y LISTO
* PARA LA RELACIÓN:
  + LA TABLA DE LA BBD DEBE ESTAR EN INNODB (SELECCIONAR LA TABLA/DARLE EL OPERACIONES/ IR A OPCIONES DE TABLA Y CAMBIAR EL MOTOR DE ALMACENAMIENTO POR INNODB)
  + TIPO INDICE = INDEX Y ESPECIFICAR LA COLUMNA QUE SE RELACIONA CON LA OTRA TABLA (LA RELACIÓN SE HACE EN LA TABLA NECESARIA)
  + SE ESCOGE ELIMINAR Y ACTUALIZAR EN CASCADA POR EL TIPO DE RELACIÓN.
  + SE ESCRIBE LOS CAMPOS RESPECTIVOS Y GUARDAR

**-------------- EJERCICIO CON NULL MYSQL Y ACCESS ----------**

* SE UTILIZA EL IS NULL SI UN CAMPO EN NULO

SELECT \* FROM CLIENTES LEFT JOIN pedidos ON clientes.`CÓDIGO CLIENTE`= pedidos.`CÓDIGO CLIENTE` WHERE `NÚMERO DE PEDIDO` IS NULL

########## SUBCONSULTAS #####################

SON CONSULTAS DENTRO DE OTRAS (UN SELECT DENTRO DE OTRO SELECT)

SUBCONSULTA ESCALONADA: Devuelve una única columna con un único registro. (se utiliza como consulta o criterio de evaluación):

SUBCONSULTA DE LISTA: Devuelve una lista de registros. AQUÍ SE UTLIZAN MUCHO LOS OPERADORES ANY, IN Y ALL

SUBCONSULTA CORRELACIONADA:

---------- EJERCICIOS SUBCONSULTA ESCALONADA -----------

* SELECT \* FROM productos WHERE PRECIO < (SELECT AVG(PRECIO) FROM productos)
* SELECT \* FROM PRODUCTOS WHERE PRECIO > (SELECT MAX(PRECIO) FROM PRODUCTOS WHERE SECCIÓN = 'CERÁMICA'

---------------- EJERCICIOS SUBCONSULTA DE LISTA ----------------

* SELECT \* FROM productos WHERE PRECIO > ALL (SELECT PRECIO FROM productos WHERE productos.SECCIÓN = "CERÁMICA")

---------- EJERICICIOS IN ------------

UNA CONSULTA QUE DEVUELVE EL NOMBRE Y PRECIO DEL ARTICULO QUE SE HAYAN PEDIDO 20 O MÁS UNIDADES

* SELECT [NOMBRE ARTÍCULO], [PRECIO] FROM PRODUCTOS WHERE [CÓDIGO ARTÍCULO] IN (SELECT [CÓDIGO ARTÍCULO] FROM [PRODUCTOS- PEDIDOS] WHERE [PRODUCTOS- PEDIDOS].UNIDADES >= 20)
* CON INNER JOIN EN VEZ DE IN:
* SELECT [NOMBRE ARTÍCULO], [PRECIO] FROM PRODUCTOS INNER JOIN [PRODUCTOS- PEDIDOS] ON PRODUCTOS.[CÓDIGO ARTÍCULO]=[PRODUCTOS- PEDIDOS].[CÓDIGO ARTÍCULO] WHERE UNIDADES >= 20

CONSULTA DONDE LOS CLIENTES NO HAN HECHO PEDIDOS O QUE LO HAN HECHO, PERO SIN PAGAR CON TARJETA

* SELECT EMPRESA, POBLACIÓN FROM CLIENTES WHERE [CÓDIGO CLIENTE]NOT IN (SELECT [CÓDIGO CLIENTE] FROM PEDIDOS WHERE [FORMA DE PAGO] = "TARJETA")
* SELECT EMPRESA,POBLACIÓN FROM CLIENTES LEFT JOIN PEDIDOS ON PEDIDOS.[CÓDIGO CLIENTE] = CLIENTES.[CÓDIGO CLIENTE] WHERE [FORMA DE PAGO] <> "TARJETA" OR PEDIDOS.[CÓDIGO CLIENTE] IS NULL

CONSULTA DONDE LOS CLIENTES HAN HECHO PEDIDOS, PERO PAGANDO SIN TARJETA

* SELECT EMPRESA, POBLACIÓN FROM CLIENTES WHERE [CÓDIGO CLIENTE] NOT IN (SELECT [CÓDIGO CLIENTE] FROM PEDIDOS WHERE [FORMA DE PAGO] = "TARJETA") AND [CÓDIGO CLIENTE] IN (SELECT [CÓDIGO CLIENTE] FROM PEDIDOS)
* SELECT EMPRESA,POBLACIÓN FROM CLIENTES LEFT JOIN PEDIDOS ON PEDIDOS.[CÓDIGO CLIENTE] = CLIENTES.[CÓDIGO CLIENTE] WHERE [FORMA DE PAGO] <> "TARJETA"

########### CONSULTAS DE ACCIÓN #############

Comando DML Y DDL

* Actualización
  + UPDATE
* Creación de tabla
  + CREATE
  + SELECT INTO
* Eliminación
  + DELETE
* Datos anexados
  + INSERT INTO

\*\*\*\* NO SE PUEDE DESHACER LAS CONSULTAS DE ACCIÓN, TOCARÍA HACER EL PROCESO INVERSO

------ UPDATE -----

UPDATE NOMBRE\_TABLA SET NOMBRE\_CAMPO=NUEVO\_VALOR

------- SELECT INTO -----

EN ESTE CASO COMO ES UNA CONSULTA DE ACCIÓN EL INTO CREARÁ UNA TABLA A PARTIR DE OTRA TABLA, NO CREARÁ UNA NUEVA DESDE CERO PORQUE ESO LO HACE LA CONSULTA DE DEFINICIÓN DE DATOS.

* FORMA EN ACCESS:
  + SELECT \* INTO CLIENTES\_MADRID FROM CLIENTES WHERE POBLACIÓN = "MADRID"
  + EL \* ES PORQUE QUEREMOS COPIAR TODAS LAS COLUMNAS
* FORMA EN MYSQL:
  + NO ACEPTA SELECT INTO, HAY QUE UTILIZAR EL CREATE TABLE
  + CREATE TABLE CLIENTES\_MADRID SELECT \* FROM CLIENTES WHERE POBLACIÓN = “MADRID”

----------- DELETE ------------

* DELETE FROM CLIENTES WHERE POBLACIÓN = “MADRID”
* DELETE FROM PRODUCTOS WHERE SECCIÓN = "DEPORTIVOS" AND PRECIO BETWEEN 50 AND 100
* Para eliminar solo un número específico de filas, puede utilizar la declaración LIMIT. El valor proporcionado LIMIT será cuántas filas afectará: Por ejemplo, esta declaración solo eliminará las primeras 5 filas que coincidan con la condición:
  + DELETE FROM table WHERE condition LIMIT 5;

###### CONSULTAS DE PREDICADO ###########

\*\*\*\* HAY QUE TENER EN CUENTA VARIOS FACTORES CUANDO AL MOMENTO DE ELIMINAR HAY TABLAS RELACIONADAS: HAY QUE TRABAJAR CON CONSULTAS DE PREDICADO.

* DISTINCT: MUESTRA SOLAMENTE UN REGISTROS, NO REPITE INFORMACIÓN ENCONTRADA POR COLUMNA
  + SELECT DISTINCT EMPRESA,POBLACIÓN FROM CLIENTES INNER JOIN PEDIDOS ON CLIENTES.[CÓDIGO CLIENTE] = PEDIDOS.[CÓDIGO CLIENTE]
* DISTINCTROW: NO REPITE INFORMACIÓN MIRANDO LAS FILAS
  + DELETE DISTINCTROW CLIENTES.\*, PEDIDOS.[CÓDIGO CLIENTE] FROM CLIENTES LEFT JOIN PEDIDOS ON CLIENTES.[CÓDIGO CLIENTE] = PEDIDOS.[CÓDIGO CLIENTE] WHERE PEDIDOS.[CÓDIGO CLIENTE] IS NULL

------------------ INSERT INTO ---------------------

* Las tablas deben tener los mismos nombres en sus columnas y deben tener el mismo tipo de datos
* Permite anexar los datos de una tabla a otra
  + INSERT INTO CLIENTES\_MADRID SELECT \* FROM CLIENTES WHERE CLIENTES.POBLACIÓN = "BARCELONA"
  + INSERT INTO CLIENTES (EMPRESA, POBLACIÓN) SELECT EMPRESA, POBLACIÓN FROM CLIENTES\_MADRID
    - ESTÁ ES LA FORMA DE INSERTA SOLAMENTE DOS COLUMNAS, SE DEBEN MANEJAR LOS MISMOS CAMPOS EN AMBAS TABLAS.
* PARA INSERTER VARIOS DATOS A LA VEZ:
  + INSERT INTO table (col1, col2, col3) VALUES (row1\_val1, row1\_val2, row1\_val3), (row2\_val1, row2\_val2, row2\_val3), (row3\_val1, row3\_val2, row3\_val3);

############# REFERENCIAS CRUZADAS ###################

\*\*\*\* MYSQL NO ACEPTA ESTE TIPO DE CONSULTAS

* UNA CONSULTA DE REFERENCIAS CRUZADAS ES AQUELLA QUE NOS PERMITE VISUALIZAR LOS DATOS EN FILAS Y EN COLUMNAS, ESTILO TABLA.
* TABLAS DINAMICAS DE EXCEL

2-ZONA / CAMPO FILAS: PARA CREARLA HAY QUE UTILIZAR PIVOT (SE PUEDEN AGREGAR LAS FILAS QUE UNO QUIERA)

3-ZONA / CAMPO COLUMNAS: PARA CREARLA HAY QUE UTILIZAR GROUP BY

1-ZONA / CAMPO TOTALES: HAY QUE UTILIZAR TRANSFORM

\*\*SIEMPRE SE EMPIEZA CON LA INSTRUCCIÓN TRANSFORM

* TRANSFORM SUM(PRECIO) AS TOTAL SELECT [NOMBRE ARTÍCULO] FROM PRODUCTOS GROUP BY [NOMBRE ARTÍCULO] PIVOT SECCIÓN
* TRANSFORM COUNT(POBLACIÓN) SELECT EMPRESA FROM CLIENTES INNER JOIN PEDIDOS ON CLIENTES.[CÓDIGO CLIENTE] = PEDIDOS.[CÓDIGO CLIENTE] GROUP BY EMPRESA PIVOT [FORMA DE PAGO]

############### COMANDOS DDL ###########################

* CREATE: Crear tabla
* ALTER: Modificar campos de tabla
* DROP: Eliminar CAMPOS, CARACTERÍSTICAS, TABLA.
* TRUNCATE: Borrar todas las filas

######### TIPOS DE TABLA ##########

* TABLAS BASE
* TABLAS DERIVADAS
* TABLAS VISTAS

######### SINTAXIS ##########

* CREATE TABLE NOMBRE\_TABLA (CAMPO1 TIPO\_DATO, CAMPO2 TIPO\_DATO…) PRIMARY KEY (CAMPO)
* Cuando intente crear una tabla con un nombre de tabla ya existente, recibirá un mensaje de error y no se modificará ni creará ninguna tabla.

########## TIPOS DE DATOS ###########

* Predefinido: Cadena, Numérico, Fecha/Hora, booleanos
* Construido:
* Definido por el usuario:

------- ALTER ---------

* ALTER TABLE NOMBRE\_TABLA ADD COLUMN NOMBRE\_COLUMNA TIPO\_DATO

----- ALTER Y PRIMERY KEY ----

* ALTER TABLE NOMBRE\_TABLA ADD PRIMARY KEY (NOMBRE\_COLUMNA)

----- ALTER Y DROP PARA ELIMINAR UNA COLUMNA ----

* ALTER TABLE NOMBRE\_TABLA DROP COLUMN NOMBRE\_COLUMNA

--- ALTER Y MODIFICAR UNA COLUMNA—

* ALTER TABLE NOMBRE\_TABLA ALTER COLUMN NOMBRE\_COLUMNA NUEVO\_TIPODATO

--- VALOR POR DEFECTO EN UN CAMPO Y QUITAR EL VALOR POR DEFECTO---

####### SET DEFAULT: ALGUNOS GESTORES DE BBDD NO ADMITEN LA INSTRUCCIÓN

* ALTER TABLE NOMBRE\_TABLA ALTER COLUMN NOMBRE\_COLUMNA SET DEFAULT VALOR\_POR\_DEFECTO
  + (EL VALOR POR DEFECTO DEPENDE DE QUE TIPO DE DATO ES EL CAMPO, POR EJ: SI ES VARCHAR ENTONCES SE ENCIERRA CON “” SI ES INT SOLAMENTE SE COLOCA EL NÚMERO)
* ALTER TABLE NOMBRE\_TABLA ALTER COLUMN NOMBRE\_COLUMNA DROP DEFAULT

------ TRUNCATE ----------

* TRUNCATE TABLE NOMBRE\_TABLA  
  SIRVE PARA BORRAR LAS FILAS DE UNA TABLA

**########### INDICES ############**

* LOS INDICES NOS PERMITEN CONSULTAR LA INFORMACIÓN DE LAS TABLAS CON MAYOR RÁPIDEZ
  + INDICE CLAVE PRIMARIA: CADA VALOR ES UNICO, NO NULL
    - ALTER TABLE NOMBRE\_TABLA ADD PRIMARY KEY (NOMBRE\_COLUMNA)
  + INDICES ORDINARIOS: PERMITE DUPLICADOS, SÍ NULL
    - CREAR UN INDICE QUE PERMITE DUPLICADOS:
    - CREATE INDEX NOMBRE\_INDICE ON NOMBRE\_TABLA (NOMBRE\_COLUMNA)
  + INDICES ÚNICOS: NO PERMITE DUPLICADOS, SÍ NULL
    - CREATE UNIQUE INDEX NOMBRE\_INDICE ON NOMBRE\_TABLA (NOMBRE\_COLUMNA)
  + INDICES COMPUESTOS: MULTIPLES COLUMNAS, SÍ NULL
    - CREATE UNIQUE INDEX NOMBRE\_INDICE ON NOMBRE\_TABLA (NOMBRE\_COLUMNA)
    - CREATE UNIQUE INDEX NOMBRE\_INDICE ON NOMBRE\_TABLA (NOMBRE\_COLUMNA1, NOMBRE\_COLUMNA2…)

**########## ELIMINACIÓN DE INDICES ##########**

* DROP INDEX NOMBRE\_INDICE ON NOMBRE\_TABLA
* PARA LLAVE PRIMARIA:
* ALTER TABLE EJEMPLO DROP CONSTRAINT NOMBRE\_INDICE\_POR\_DEFECTO
* EN MYSQL SOLO BASTA CON ESCRIBIR ALTER TABLE NOMBRE\_TABLA DROP PRIMARY KEY

######### TRIGGERS (DISPARADORES) #########.

* Tiene que estar asociado a una tabla porque desencadena una acción cuando ocurra algo en esa tabla (insertar, actualizar, eliminar)
* SE UTILIZAN PARA TAREAS DE MANTENIMIENTO Y ADMINISTRACIÓN EN BBDD
* EN ACCESS NO SE PUEDE UTILIZAR, PERO TIENE MACROS QUE SIMULAN ALGO PARECIDO

######### CUANDO SE EJECUTAN LOS TRIGGERS #####

* SE PUEDEN EJECUTAR ANTES O DESPUÉS DE ELIMINAR, ACTUALIZAR O INSERTAR

------- EJEMPLO TRIGGER CON INSERT Y AFTER

* CREATE TRIGGER PRODUCTOS\_AI AFTER INSERT ON productos FOR EACH ROW INSERT INTO reg\_productos(CÓDIGOARTÍCULO,NOMBREARTÍCULO,PRECIO,INSERTADO) VALUES (NEW.`CÓDIGO ARTÍCULO`, NEW.`NOMBRE ARTÍCULO`, NEW.PRECIO, NOW())
* LOS NEW HACEN REFERENCIA A LOS CAMPOS DE LA TABLA productos
* La nomenclatura del nombre se los triggers es el nombre de la tabla con la que esta relacionada, el a o b si es after o before y la acción i,d,u si se va a insertar, eliminar o actualizar.

######## TRIGGER DE ACTUALIZACIÓN #######

* SIRVE COMO UN BACKUP (TABLA DE RESPALDO DE LOS REGISTROS ANTES DE SU MODIFICACIÓN – INSTRUCCIÓN OLD)
* CREATE TRIGGER TABLA\_BU BEFORE UPDATE ON productos FOR EACH ROW INSERT INTO reg\_productos(CÓDIGOARTÍCULO,NOMBREARTÍCULO,PRECIO,USUARIO) VALUES (OLD.`CÓDIGO ARTÍCULO`, OLD.`NOMBRE ARTÍCULO`, OLD.PRECIO)

########### TRIGGER DE ELIMINACIÓN ####

* CREATE TRIGGER TABLA\_AD AFTER DELETE ON productos FOR EACH ROW INSERT INTO reg\_productos(CÓDIGOARTÍCULO,NOMBREARTÍCULO,PRECIO,USUARIO) VALUES (OLD.`CÓDIGO ARTÍCULO`, OLD.`NOMBRE ARTÍCULO`, OLD.PRECIO)

###### ELIMINAR UN TRIGGER ######

* COMO NO EXISTE UNA INSTRUCCIÓN CON ALTER ENTONCES TOCA ELIMINAR EL TRIGGER Y VOLVERLO A CREAR CON EL DROP Y CREATE
* DROP TRIGGER NOMBRE\_TABLA\_TRIGGER
* UTILIZAR OTRA VEZ LA INSTRUCCIÓN CREATE TRIGGER PARA CREAR EL TRIGGER
* PARA HACER TODO ESTO EN UNA INSTRUCCIÓN SERÍA:
  + DROP TRIGGER IF EXISTS NOMBRE\_TABLA\_TRIGGER; CREATE TRIGGER … (CONTINUAMOS CON TODA LA INFORMACIPON PARA CREAR UN CREATE TRIGGER)

######### PROCEDIMIENTO ALMECENADO #########

* ALMACENAR EN UN PRODECIMIENTO LAS INSTRUCCIONES REPETITIVAS. (EFICIENCIA Y SEGURIDAD)
* ACCESS NO ACEPTA PROCEDIMIENTO ALMACENADO
* SINTAXIS:
  + CREATE PROCEDURE NOMBRE\_PROCEDIMIENTO(PARAMETROS OPCIONALES)

INSTRUCCIÓN SQL…

POR EJEMPLO:

* + CREATE PROCEDURE MUESTRA\_CLIENTES()

SELECT \* FROM CLIENTES WHERE POBLACIÓN = “MADRID”

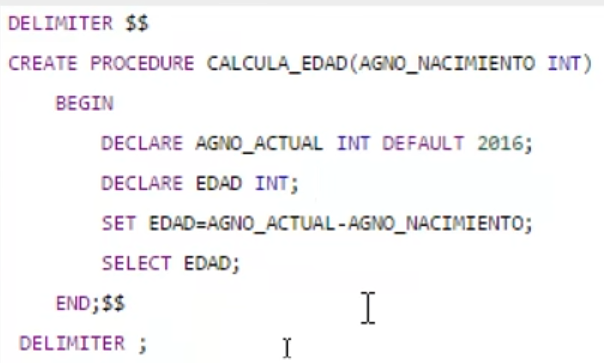
* LLAMADA AL PROCEDIMIENTO:
  + CALL NOMBRE\_PROCEDIMIENTO()
* LLAMADA CON PARAMETROS:
  + CREATE PROCEDURE ACTUALIZA\_PRODUCTOS (N\_PRECIO INT, CODIGO VARCHAR(4))
  + UPDATE PRODUCTOS SET PRECIO=N\_PRECIO WHERE CÓDIGOARTÍCULO=CÓDIGO
* SI EL PROCEDIMIENTO TIENE MÁS DE UNA LINEA DE CÓDIGO ENTONCES HAY QUE COLOCAR LAS INSTRUCCIONES BEGIN Y END
  + CREATE PROCEDURE NOMBRE()

BEGIN

…

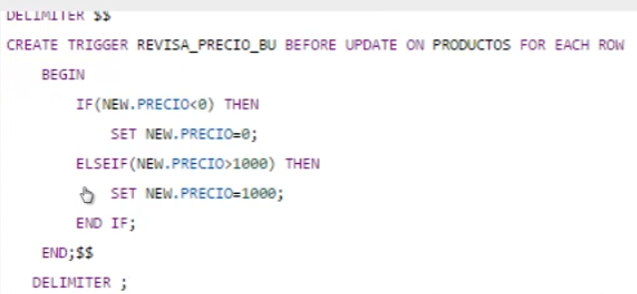
END

* DECLARAR VARIABLES:
  + DECLARE NOMBRE\_VARIABLE TIPO\_DATO
  + SI ES UNA CONSTANTE ENTONCES DECLARE NOMBRE\_VARIABLE TIPO\_DATO DEFAULT VALOR



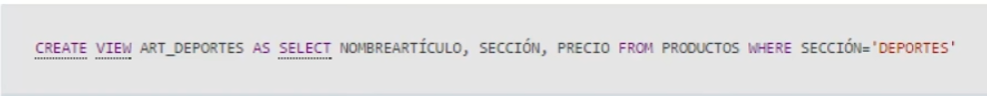
* + EL SELECT FUNCIONA COMO EL RETURN
  + DELIMITER $$ ES EL DELIMITADOR POR BLOQUE PARA SABER CUANDO TERMINA NUESTRO PROCEDIMIENTO

##### TRIGGER CON PROCEDIMIENTO #####



#### VISTAS (VIEWS) ###

* VENTAJAS:
  + PRIVACIDAD DE LA INFORMACIÓN.
  + OPTIMIZACIÓN DE LA BBDD.
  + ENTORNO DE PRUEBAS.



SI SE HACE UN MODIFICACIÓN EN LA TABLA ORIGINAL ENTONCES LA VISTA TAMBIÉN SE VERÁ AFECTADA, CAMBIARÁ.

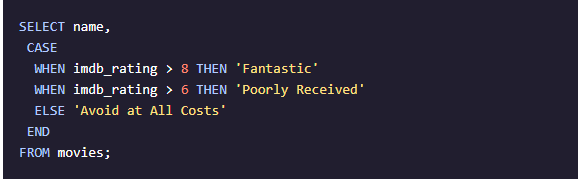
* PARA ELIMINAR: DROP VIEW ART\_DEPORTES

PARA MODIFICAR LA VISTA:



########## CASE ############

* Una CASE declaración nos permite crear diferentes salidas (generalmente en la SELECT declaración). Es la forma en que SQL maneja la lógica si-entonces.
* CASE crea diferentes salidas.
* UTILIZAMOS LA INSTRUCCIÓN WHEN Y EL THEN



SELECT name,

  CASE

    WHEN genre = 'romance' THEN 'Chill'

    WHEN genre = 'comedy' THEN 'Chill'

    ELSE 'Intense'

  END AS 'Mood'

FROM movies;