PARTE PRÁCTICA:
SQL

Cuando creemos tablas en SQL empezamos creando una “Database” con el comando Create Database if not Exists “Nombre de la Database” y continuamos con un Use “Nombre de la Database” para darle uso.

Una vez la tenemos, empezamos creando tablas con Create Table if not Exists “Nombre de la Tabla” , y abrimos paréntesis para poner los atributos (.

Cuando añadimos atributos empezamos con el “Nombre del Atributo” y continuamos poniendo su tipo y sus límites CHAR (), VARCHAR(), INT, DOUBLE, DATE, ENUM… , una vez puesto declaramos si es Null o Not Null (Si la tabla tiene una única Primary Key la pondremos aquí en vez de los Nulls) y acabamos con una “,”.

Si tuvieran varias Primary Key se pondrían al final de la tabla tal que así “Primary Key (Nombre de la Primary Key 1, Nombre de la Primary Key 2...)”.

Al rematar con los atributos, declaramos las varias Primary Key (si las hay) y las Foreign Key que sean heredadas de otras tablas, para declarar las Foreign Key hacemos así.

Foreign Key (“Nombre de la Foreign Key”) References “Nombre de la Tabla Referenciada” (“Nombre de la Foreign Key en la Tabla Referenciada”)

Acabamos la tabla cerrando el paréntesis “)” y con un punto y coma “;”.

Para añadir nuevas Claves Foráneas más tarde, se usa el comando Alter Table “Nombre de la Tabla” + Add Constraint “Nombre de la Clave Foránea” , esto se usa en el caso de que no podamos crear la tabla previamente debido a que tengan una clave foránea la una con la otra y viceversa. (También podemos crear una tabla aparte como hacíamos con las relaciones N,M en el modelo Entidad Relación.)

(“Add Constraint” no vale solo par claves foráneas, si no que con este comando podemos añadir cualquier tipo de dato a nuestras tablas)

Es MUY IMPORTANTE crear las tablas y relaciones en orden y teniendo en cuenta ciertas normas, empezaremos siempre por las Tablas Independientes, o sea, las que no tengan Foreign Keys, y a partir de ahí creamos las referenciadas, ya que si no SQL no será capaz de hacer una referencia de una tabla aún no creada.

Recordad que las flechas son importantes, las relaciones 1,N nos dan la pista de en qué dirección funcionan las Foreign Key (La 1 es de donde sale y la N es donde termina)

Para hacer relaciones Reflexivas, creamos una Foreign Key de la misma columna y ya está, es lo obvio.

No es necesario, pero podemos poner Engine = InnoDB entre la paréntesis final “)” y el punto y coma “;” de una tabla, pero SQL ya lo declara por defecto.

Para hacer comentarios o enumeraciones en los ENUM usamos comillas ‘ ‘ , si lo hemos hecho bien, el resultado entre ellas debería aparecer en VERDE.

El comando “SELECT” como su nombre indica, nos ayuda a hacer selecciones completas de un listado que queramos según los parámetros que definamos.

“DISTINCT” es un atributo que evita que se repitan datos de las columnas cuando se nos muestran, por ejemplo “SELECT DISTINCT” nos enseñará solo los datos únicos.

El comando “AS” nos sirve para darle un alias a una selección, o sease, un apodo con la que la podamos identificar de forma más sencilla.
“SELECT (Nombre_de_la_Fila) AS (La_voy_a_llamar_así)”

“ORDER BY” es exactamente lo que nos esperaríamos, nos sirve para ordenar datos de columnas siguiendo los criterios que nosotros marquemos.
“select * from clientes order by codrepcliente”

“DESC” ordena nuestros datos de mayor a menor.

“ASC” ordena nuestros datos de menor a mayor.

/* ASC y DESC van después de la función a la que acompañan */

“WHERE” es un comando condicional que nos ayuda a establecer los parámetros en una búsqueda, para ello nos ayudamos de los símbolos “>”, “>=”, “<”, “<=”, “=”, “<>”, “!=”.

SELECT nombre, oficina, feccontrato FROM empleado WHERE oficina = 12 AND YEAR (fecContrato) = 1990;

“AND && OR || NOT” son usados tal cual los usaríamos en java.

SELECT * FROM empresasabc, empleado;
SELECT nombre, sueldo
FROM empleados

WHERE sueldo >= 2000 && sueldo <= 2500

“BETWEEN” es otro condicional bastante obvio que nos permite seleccionar datos entre dos puntos concretos.
“IN” es otro condicional que nos indica exactamente una lista de valores que tienen que ser los que se nos muestren de la tabla.

SELECT nombre, sueldo, oficina
FROM empleados
WHERE region IN (1, 2, 3, 4, 5)

“IS” o “IS NOT” pueden usarse para varias cosas, (también como condicionales) para indicar si un dato tiene que estar en o fuera de un rango o una característica.

“NULL” nos indica si algo puede ser nulo o no.

SELECT nombre, oficina
FROM empleados
WHERE oficina IS NOT NULL;

IF nos ayuda con muchas funciones y como condicional, si por ejemplo, lo usamos para comprobar un boolean (True , False).

IF (Dato 1 = Dato2, ‘True’, ‘False’)

/* Si el statement es verdadero, nos escribe lo que pongamos detras de la primera coma, en caso contrario, se escribirá lo de la segunda. */

Los comodines nos sirven como pues eso… comodines para substituir ciertas preguntas o datos según su significado.

% Muchos caracteres

_ Un único carácter

? Un único carácter en Windows
“LIKE” tiene el mismo significado que el símbolo “=” pero se usa en substitución de este. Se puede usar junto a NOT haciendo un “NOT LIKE” que sería algo parecido a esto “!=”

```
SELECT nombre, oficina
FROM empleados
WHERE oficina  LIKE ‘A%’;
```

```
SELECT nombre, oficina
FROM empleados
WHERE oficina  LIKE ‘P_erez’;
```

“RLIKE” nos sirve para indicar un patrón entre un punto y otro.

```
SELECT nombre, oficina
FROM empleados
WHERE oficina  LIKE ‘’;
```

“LEFT” y “RIGHT” nos sirven para recoger caracteres desde la derecha o la izquierda además, le podemos añadir el número de caracteres que queramos que recoja.

De la misma forma “MID” recoge los caracteres del medio.

```
SELECT LEFT (5) nombre, oficina
FROM empleados;
```

```
SELECT nombre, fecNacimiento,
YEAR (CURDATE())-YEAR(FecNacimiento) -
IF(RIGHT(fecNacimiento), 5) > RIGHT ((CURDATE(), 5), 1, 0)
```

El operador “UNION” - “UNION ALL” nos sirve para unir dos selecciones diferentes y transformarlas en una misma selección compartida que enlaza los criterios de ambas partes.

```
SELECT idFabricante, idProducto
FROM Productos
WHERE existencias < 50
UNION
SELECT Fabricante, producto, cantidad
FROM LineasPedido
WHERE cantidad between 30 and 40;
```

“JOIN” y “ON” une dos tablas que compartan parámetros iguales entre ellas. (Los modificadores “LEFT” y “RIGTH” nos ayudan a escoger cual de las tablas tendrá preferencia, si la que está a la derecha del “JOIN” o a la izquierda.)

```
SELECT codcliente, nombre, codpedido, fechapedido
FROM clientes JOIN pedidos
ON clientes.codcliente = pedidos.codcliente;
```

Con “USING” escogemos una tabla y la seleccionamos para ser usada por ciertos parámetros concretos y acciones de la lista de comandos de SQL.

Con “GROUP BY” + “nombre de la columna” nos ayuda a hacer grupos y agrupar la información de las columnas.

Con “HAVING” + “condición” ponemos una condición, es básicamente igual que “WHERE”, pero se usa después de una función de agregado.(?)

```
Having Avg(Sueldo) > 3
```

Con “LIMIT” + ponemos un límite al número de filas que queremos mostrar/buscar por pantalla.

- Los Nombres Cualificados “Nombre de la Tabla + alias”, “Nombre de campo + alias” nos sirven para hacer consultas donde los datos de las tablas están situados en tablas diferentes, los alias son “motes” temporales que sirven para identificar una tabla de otra.

Ejemplo: Pedidos p, p.codCliente.

Con “INNER JOIN” hacemos algo parecido a lo que hacemos con los nombres Cualificados, lo único que cambia es el método de hacerlo. Está función tiene dos modificadores, “Left” y “Right Join”, estes modificadores nos permiten ver o no ver las opciones de la selección que tengan “Null” como valor.

Funciones de agregado:

SUM(expresion) = Sumar

AVG(expresion) = Media

MIN(expresion) = Mínimo

MAX(expresion) = Máximo

COUNT(expresion) = Contar

COUNT(*) = Cuenta valores distintos a null

COUNT(distinct expresion) = Cuenta valores distintos

VARIANCE (expresion) = Varianza

TRANSACCIONES

Las transacciones solo se realizan en el caso de la Modificación de las tablas, las funciones select no cambian nada en las tablas, por lo tanto no las usan, estas solo se dan en el caso de que haya una alteración en las tablas múltiples.

Ejemplo:

DEVOLUCIÓN:

```
INSERT INTO Devoluciones VALUES (1, 1500, NOW())
UPDATE libros SET Prestado SINo = TRUE WHERE codlibro = 1500;
```

Los cambios realizados en las Transacciones solo se realizan en teoría, no pasan a ser cambios reales dentro de las tablas a no ser que usemos el comando “COMMIT”.

El comando “ROLLBACK” es una especie de deshacer “Ctrl - Z” que no sirve para volver al estado anterior de una declaración que hayamos hecho.

El comando “SAVEPOINT” nos sirve para guardar las consultas SQL hasta un punto que nosotros deseemos dentro de la propia consulta, dejándolas en el estado en el que se quedaron entonces, “ROLLBACK TO SAVEPOINT” recupera la consulta en el estado en el que estaba en el “SAVEPOINT” que hayamos declarado.

Podemos cambiar el estado de SQL a un estado “AUTO-COMMIT”, esto quiere decir que todos los cambios que hagamos serán considerados como si tuvieran un “COMMIT” al final y se apliquen nada más declararlos. /* SET GLOBAL autocommit = 0; */

StaRT TRANSACTION

COMMIT

ROLLBACK

SAVEPOINT

Para evitar que los diferentes usuarios de una Base de Datos alteren o no las tablas se usan los comandos “LOCK” o “UNLOCK TABLE”, muy básico.

CREATE VIEW mbVista AS
SELECT...

EJEMPLO DE TRANSACCIÓN COMENTADO Y EXPLICADO. (Ctrl - C + Ctrl - V en SQL).

```
/* Escogemos la DataBase que queremos usar para el ejercicio. */

USE empresasabc;

/* Inicializamos la transacción. */

START TRANSACTION;

/* EJERCICIO: El cliente Antonio Canalesnos realiza el pedido 220 en el día de hoy de los siguientes productos:
Producto Cantidad Precio de Venta Wii Cable AV DRAGON 310 € Nokia C5-014 220€ TDT Portatil 3.5 Intenso TV Star 695€ El comercial que realiza el pedido es
 Baqueira Jaumes, Alvaro. Realizar las operaciones necesarias para guardar en nuestra base de datos el pedido anterior. Hay que tener en cuenta
 que la comisión del vendedor es 10% del importe total de la factura; las existencias de los productos disminuye y las ventas de la oficina donde
 trabaja el vendedor aumentan. */
```

/* Comenzamos el ejercicio haciendo un INSERT en la tabla Pedidos, ya que el ejercicio nos pide que sea así, dentro de la tabla, seleccionamos todos los datos que tenemos. Los que no (CodCliente y CodRepresentante) los tenemos que sacar a través de subconsultas dentro del propio
SELECT para sacar el código del empleado y del cliente de sus respectivas tablas. */

INSERT INTO Pedidos (codPedido, fechaPedido, codCliente, codRepresentante) VALUES
(223, CURDATE(), (SELECT codCliente FROM Clientes
WHERE nombre = 'Antonio Canales'), (SELECT codEmpleado FROM Empleados
WHERE nombre = 'Baqueira Jaumes, Alvaro'));

/* Una vez hemos creado los datos del pedido, debemos sacar los datos de los productos que están en él, una vez más, seleccionamos aquellos que ya tenemos o que nos han dado en el enunciado del ejercicio, los que no tenemos (IdFabricante e IdProducto) los tenemos que volver a sacar
de sus correspondientes Subconsultas, usando la descripción del producto como pista para encontrar los datos.

Todo esto rellenando siempre todos los datos de la tabla en la que estamos. Tenemos que hacer esto con los 3 productos que se nos dan en el
ejercicio.*/

/* Si nos preguntamos por qué estamos usando la tabla LineasPedido en vez de la propia de Productos, hay que recordar las lecciones del 1er Trimestre, ya que esta tabla está creada especificamente para no tener que repetir los datos de los Productos 3 veces y rellenar 3 veces la misma
tabla.*/

```
INSERT INTO LineasPedido (codPedido, numLinea, fabricante,
producto,
cantidad, precioVenta) VALUES

(223, 1, (SELECT idFabricante FROM Productos WHERE
descripcion = 'Wii Cable AV DRAGON'),
(SELECT idProducto FROM Productos WHERE
descripcion = 'Wii Cable AV DRAGON'), 3, 10),

(223, 2, (SELECT idFabricante FROM Productos WHERE descripcion =
'Nokia C5-01'),
(SELECT idProducto FROM Productos WHERE descripcion = 'Nokia
C5-01'), 4, 220),

(223, 3, (SELECT idFabricante FROM Productos WHERE descripcion =
'TDT Portatil 3.5 Intenso TV Star'),
(SELECT idProducto FROM Productos WHERE descripcion = 'TDT
Portatil 3.5 Intenso TV Star'), 6, 95);

/* Cuando acabamos de rellenar los datos, tanto del cliente y el
empleado como de los productos y la venta, es hora de registrar los
cambios
que estas acciones han tenido en el resto de tablas, es decir: */

/* Debemos aumentar la comisión del empleado 'Baqueira Jaumes,
Alvaro' tras la venta tal y como nos pone el ejercicio */

UPDATE Empleados SET Comision = IFNULL(Comision, 0) +
(SELECT SUM(cantidad * PrecioVenta)*0.1
FROM LineasPedido WHERE codPedido = 223)
WHERE nombre = 'Baqueira Jaumes, Alvaro';

/* Debemos quitarle la cantidad de productos vendidos al stock que
tenemos registrado en la tabla de Productos.*/
```

```
UPDATE Productos SET Existencias = IFNULL(existencias, 0) - 3
WHERE descripcion = 'Wii Cable AV DRAGON';

UPDATE Productos SET Existencias = IFNULL(existencias, 0) - 4
WHERE descripcion = 'Nokia C5-01';

UPDATE Productos SET Existencias = IFNULL(existencias, 0) - 6
WHERE descripcion = 'TDT Portatil 3.5 Intenso TV Star';

/* Debemos poner la cantidad de dinero ganado por la oficina a la que
pertenece el empleado 'Baqueira Jaumes, Alvaro' tras la venta.*/

UPDATE Oficinas SET Ventas = IFNULL(Ventas, 0) +
(SELECT SUM(cantidad * PrecioVenta) FROM LineasPedido WHERE
codPedido = 223)
WHERE codOficina = (SELECT oficina FROM Empleados WHERE
nombre = 'Baqueira Jaumes, Alvaro');

/* Todo esto se tiene que ver a través de la lógica y relacionando las
tablas como corresponde según las flechas que las unen. (La función
IFNULL se usa en los casos anteriores para evitar errores en el caso de
que el valor de cualquier dato fuera 0). */

/* Una vez acabamos las transacciones, ejecutamos el código, si vemos
que no nos da fallos y que efectivamente, la venta se ha realizado con
éxito, usamos COMMIT para hacer los cambios PERMANENTES; en el
caso contrario, la función ROLLBACK nos permite deshacer
absolutamente todos los
cambios de la transacción y empezar de cero para corregir posibles
fallos. */

COMMIT;

ROLLBACK;
```

FUNCIONES DE CADENAS

ASCII nos permite ver el código ASCII de un carácter concreto.

CHAR

LENGTH nos dice cual es la cantidad de caracteres que contiene un String.

REPEAT nos deja

COMPRESS hace que

CONCAT une columnas, strings o textos y los imprime de tal forma que se vean juntos.

UPPER y LOWER nos sirven para poner en mayúscula o minúsculas los datos que seleccionemos (También nos vale UCASE y LCASE).

CONCAT_WS es lo mismo que concat pero nos da la opción de escoger algo (la primera línea de código que pongamos) que se repita en lugar de la barra espaciadora.

INSERT nos inserta un String que queramos dentro de una selección de esta manera (‘Lo que queramos substituir’, ‘Numero del caracter del que partimos’, ‘Numero de caracteres que substituimos’, ‘Tecto que queramos insertar’)

```
SELECT INSERT('Estamos en clase', 4, 6, 'XXXXXX');
```

/* Aquí insertaría el texto ‘XXXXXX’ a partir del 4 caracter de Estamos en clase y borrando 6 caracteres hacia adelante. */

INSTR busca dentro de las tablas cuantas veces se repite y donde está un String concreto que queramos.

```
SELECT INSTR(nombre,  '  ') FROM empleados;
```

/* Aquí nos indicaría los espacios en blanco en la tabla nombre dentro de la tabla empleados */

LOCATE es exactamente igual que INSTR, pero INSTR siempre va a empezar en la primera posición, mientras que LOCATE puede empezar en la que se le indique. (LOCATE por defecto, solo coge la primera vez que aparece el carácter buscado “Si buscaramos ‘A’ en “Ana María”, solo aparecería la ‘A’ inicial.)

```
SELECT nombre, LOCATE(' ', nombre, 1) FROM empleados;
```

/* Está localizando los espacios en blanco de los nombres de la tabla empleados */

TRIM, LTRIM, RTRIM Son funciones que quitan los espacios en blanco o por los dos lados, o por la izquierda, o por la derecha.

REPLACE tiene la misma funcion que INSERT pero es mas especifico en la seleccion, ya que le indicamos los parametros concretos de la busqueda.

```
SELECT REPLACE(nombre, 'e', 'XXXXXX') FROM Empleados;
```

/* Aquí substituye todas las ‘e’s por XXXXXX, así de simple. */

REVERSE le da la vuelta a un String, así de simple.

SPACE es tan básico como indicar un número de espacios en blanco que queramos poner, es casi lo mismo que pulsar varias veces la Barra Espaciadora.

SUBSTRING o SUBSTRING_INDEX nos permite extraer un String o una parte de un String

```
SELECT SUBSTRING('hola, buenos días y callaros por favor', 5, 10);

/* Aquí arranacará desde el 5º caracter y cogerá 10 caracteres del String, o sea 'buenos día' */
```

SUBSTRING_INDEX te pilla toda la selección que escojas desde un parámetro, hasta el límite de ese parámetro.

```
SELECT SUBSTRING_INDEX(nombre, ' ', 2) FROM Empleados;

/* Aquí coge todos los  nombres hasta llegar al 2º espacio en blanco. */
```

FUNCIONES DE CONVERSIÓN DE NÚMEROS

Sirven para pasar tipos de caracter a otros tipos (Si es posible) y para ayudar con funciones numéricas.

BIN() Pasa datos a binario

CONV() Convierte números entre distintas bases.

HEX() Conversión de números a hexadecimal.

vOCT() Convierte un número a Octal.

UNHEX() Convierte números de hexadecimal a ASCII.

Aunque ya explicamos la función IF anteriormente, podemos usarla con las variables IFNULL y NULLIF.

Usamos IFNULL en el caso de que la expresión con las que estamos jugando tenga un valor “null”.

Sin embargo, en NULLIF nosotros declaramos que la función de la expresión tenga el valor “null”.

CASE funciona igual que un Switch de Java, declaramos una serie de casos y condiciones en las que si estas se cumplen, imprimimos la respuesta adecuada.

CAST o CONVERT son funciones muy útiles, que literalmente nos dejan pasar números o caracteres a formatos con los que sean compatibles.

SELECT ‘Juana María’, CAST (‘Juana María’ AS ASCII)

COALESCE

RAND crea un número aleatorio.

EL EXAMEN DE ESTE TRIMESTRE ES HASTA AQUÍ.

(Apunte importante, si hacemos empezar alguna de las funciones lectoras desde el fina, -1 cuenta como si fuera el final del String.)