U.T. IV.- Scripts, Índices y Vistas.

1.- Scripts.

- Desde el punto de vista del lenguaje SQL, un script es un conjunto de comandos y sentencias **SQL** en formato de texto plano que se ejecutan en una instancia del servidor de BBDD.
- En MySQL todo comando o sentencia debe terminar en un punto y coma.
- Debemos tener en cuenta que SQL es un lenguaje interpretado, es decir, las sentencias son traducidas y ejecutadas una a una en el mismo orden en el que encuentran en el script, excepto en el caso de encontrarse con una sentencia de control del flujo de ejecución.
- MySQL considera que se ha terminado una sentencia cuando se encuentra con un punto y coma.

Ejercicio 401.

Crear un script con tres sentencias:

Abrir la BD Compraventa.

Visualizar el contenido de la tabla EMPLEADOS.

Mostrar la fecha actual del sistema.

Guardar el script en un fichero.

Abrir el fichero y ejecutarlo.

- Puede modificarse el carácter de fin de sentencia con delimiter.

Ejercicio 402.

Igual que Ejercicio 401 sustituyendo el carácter de fin de sentencia por //.

2. Variables de Usuario.

Manual: *9.3. Variables de usuario*

- MySQL soporta variables que permiten almacenar un valor y hacer referencia a él posteriormente; esto posibilita pasar valores de una sentencia a otra.
- MySQL distingue entre 2 tipos de variables:
  - Variables de usuario.
  - Variables locales.
    - Se utilizan dentro de procedimientos almacenados.
- Las variables de usuario son específicas de la conexión.
  - Es decir, una variable definida por un usuario no puede ser vista o utilizada por otros y desaparece cuando el usuario abandona la conexión.
- Las variables de usuario pueden emplearse en cualquier parte donde se permitan expresiones.
- Las variables de usuario en MySQL no se declaran explícitamente ni se les asigna un determinado tipo, sino que son inicializadas al mismo tiempo que se declaran y su tipo será el del valor que contengan en cada momento.
  - Una variable de usuario puede tomar un valor Null.
- El nombre de una variable de usuario debe comenzar por el símbolo **@**.
- Una variable de usuario se declara e inicializa mediante la siguiente sentencia:

  SET|SELECT {@nombrevariable1:=expresión1} [,...]

  - Si se utiliza la opción SELECT, además de crear e inicializar una o más variables se muestra su contenido.

Ejercicio 403.

a) Declarar una variable de usuario asignándole un número entero. Visualizar todos los registros de la tabla PRODUCTOS cuyo precio unitario sea inferior al contenido de la variable.

b) Almacenar en una variable de usuario el nombre del empleado cuyo apellido es 'Fuller'. Visualizar el contenido de la variable.

c) Visualizar el nombre de todos los productos que han sido pedidos alguna vez en cantidad superior a 10 y con un 15% de descuento. Almacenar previamente dichos valores en sendas variables de usuario.

3.- Control del flujo de ejecución en un script.

Manual: *12.2. Funciones de control de flujo*

- Al contrario de lo que sucede con otros gestores de BBDD como SQL SERVER, MySQL no permite la utilización de sentencias **IF** o **WHILE** directamente en un script si no van contenidas dentro de un procedimiento almacenado.
- Para controlar el flujo de ejecución en un script sólo pueden utilizarse sentencias **CASE** y funciones **if()**.

3.1.- Sentencia CASE.

- Una sentencia CASE puede utilizarse en cualquier lugar de una sentencia SQL que admita el valor devuelto por CASE.
- Sintaxis 1:

```
CASE valor {WHEN valor_comparación_1 THEN resultado_1} [...]
[ELSE resultado_n]
END
```

Ejemplo:

```
set @ecivil=1;
select case @ecivil
    when 0 then 'soltero'
```

```
        when 1 then 'casado'
        else 'otro'
    end
    as estadocivil;
```

- Sintaxis 2:

```
CASE {WHEN expresión_lógica_1 THEN resultado_1} [...]
[ELSE resultado_n]
END
```

Ejemplo:

```
set @permiso:=false,@coche:=false;
select case when not @permiso then 'No tiene permiso'
when @coche then 'Tiene permiso y coche'
else 'Tiene permiso pero no tiene coche'
end
as Transporte;
```

- En cualquier caso, si no se especifica la cláusula ELSE, devuelve **NULL** si no hay ningún resultado coincidente.

Ejercicio 404.

a) Sentencia que, por cada fila de la tabla CLIENTES, muestre lo siguiente:

NombreCompañía, NuevoCargoContacto, AntiguoCargoContacto

El NuevoCargoContacto será el mismo que el antiguo excepto si es 'Agente de ventas', que ahora será 'Comercial' y si es 'Asistente de marketing', que ahora será 'Asistente comercial'.

b) Script que haga lo siguiente:

Almacene en una variable el valor total del pedido 10248, sin aplicar los descuentos.

Almacene en una variable el descuento a aplicar, dependiendo del valor del pedido, atendiendo al siguiente baremo:

0% si el valor del pedido es menor que 50, 5% si está entre 50 y 200 y 10% si es mayor que 200.

Visualice el valor del pedido sin y con descuento.

3.2.- Función if().

- Sintaxis:

  if(expr-lógica,expr1,expr2);

  - Si expr-lógica es cierta se ejecuta expr1 y si es falsa se ejecuta expr2.
  - Las funciones if() pueden anidarse, de manera que expr1 y/o expr2 pueden ser a su vez funciones if()
- La función if() retorna un valor numérico o una cadena de caracteres dependiendo del contexto en el que se ejecuta.
- Debe utilizarse dentro de sentencias SET ó SELECT

Ejemplo:

```
a) set @ecivil=1;
   select if(@ecivil=0,'soltero','casado') as estadocivil;
b) set @ecivil=1;
   select if(@ecivil=0,'soltero',if(@ecivil=1,'casado','otro')) as estadocivil;
```

Ejercicio 405.

a) Sentencia para obtener el mismo resultado que el del ejercicio 404 a) utilizando una función if() en vez de una sentencia CASE.

b) Idem para el ejercicio 404 b)

4.- Transacciones.

Manual: *13.4. Comandos transaccionales y de bloqueo de MySQL*

- Si queremos que un conjunto de sentencias SQL se ejecuten en conjunto, es decir, que o todas tengan éxito o todas fallen, debemos hacer uso de una "Transacción".
- Una transacción agrupa un conjunto de sentencias dentro de un bloque que, o se ejecuta en su totalidad o, en caso

de fallo inesperado, se deshacen las sentencias que ya se hubieran ejecutado.

  - De esta forma se garantiza que no haya *una* actualización *parcial* de la base de datos.

- Una transacción debe cumplir la denominada "Prueba ACID:
  - **A**tomicidad: si todas las operaciones tienen éxito, los cambios se confirman en la base de datos. Si cualquiera de las operaciones falla, toda la transacción se deshace. Es decir, no hay una actualización parcial.
  - **C**onsistencia: una transacción transforma la base de datos de un estado consistente a otro estado consistente.
  - a**I**slamiento: los cambios en una transacción no son visibles para otra transacción.
  - **D**urabilidad: los cambios son duraderos y nunca se pierden.
- MySQL incorpora un tipo de motor de base de datos (InnoDB) que permite trabajar con transacciones.
  - Manual: *15. Motores de almacenamiento InnoDB.*
- Para comprobar si nuestro gestor de BD soporta transacciones podemos ejecutar la instrucción

```
show variables like '%innodb%';
```

y comprobar que la variable have_innodb tiene el valor YES.

4.1.- AUTOCOMMIT, COMMIT y ROLLBACK

- Algunos SGBD como, MySQL (motores InnoDB o BDB) funcionan por defecto en modo **AUTOCOMMIT**. Esto significa que la ejecución de cada comando SQL se

confirma automáticamente y que los efectos/cambios del comando no podrán ser deshechos.

- Es decir, en el momento en el que se ejecuta satisfactoriamente una sentencia de actualización en una tabla (**INSERT**, **UPDATE** o **DELETE**), el resultado queda reflejado en el disco.

- Si queremos desactivar el modo **AUTOCOMMIT** para que debamos confirmar los cambios antes de que se realicen deberemos escribir el comando

SET AUTOCOMMIT=0;

- Para volver a una confirmación inmediata escribiremos

SET AUTOCOMMIT=1;

- Para ver el contenido de la variable que le indica al sistema en qué modo se encuentra:

SELECT @@AUTOCOMMIT;

- Una vez desactivado el modo **AUTOCOMMIT**, si queremos almacenar los cambios en disco debemos indicarlo mediante el comando

COMMIT;

- Si queremos deshacer los cambios realizados hasta el momento escribiremos el comando

ROLLBACK;

- La ventaja del comando **ROLLBACK** es que cuando la lógica de la aplicación programada en la transacción no puede completarse, no es necesario realizar una serie de operaciones de vuelta atrás comando a comando, sino que el trabajo puede cancelarse completamente mediante este comando.

- Aquellas transacciones no confirmadas al final del programa debido a fallos en la aplicación, fallos del sistema o en caso de conflictos por concurrencia pueden ser automáticamente deshechas.

- Para comprobar si el resultado de ejecutar una sentencia de actualización ha quedado reflejado en disco o no, debemos salir de la instancia y volver a entrar.

Ejercicio 406.

¿Cuál es el resultado de ejecutar cada uno de los siguientes scripts?

a)
```
USE compraventa;
INSERT INTO clientes(idcliente) VALUES('a');
```

b)
```
USE compraventa;
SET AUTOCOMMIT=0;
INSERT INTO clientes(idcliente) VALUES('a');
```

c)
```
USE compraventa;
SET AUTOCOMMIT=0;
INSERT INTO clientes(idcliente) VALUES('a');
COMMIT;
```

d)
```
USE compraventa;
SET AUTOCOMMIT=0;
INSERT INTO clientes(idcliente) VALUES('a');
ROLLBACK;
```

4.2.- START TRANSACTION

- Podemos ver las ventajas de funcionar en modo "transacción" mediante el ejemplo de la transferencia bancaria, supongamos estas 2 instrucciones de actualización de datos:

```
UPDATE Cuentas SET Saldo=Saldo - Cantidad WHERE IdCliente='cliente1'
UPDATE Cuentas SET Saldo=Saldo + Cantidad WHERE IdCliente='cliente2'
```

- Si, por cualquier circunstancia, se ejecutara la primera sentencia y no se ejecutara la segunda, se produciría una inconsistencia en la BD.
- El resultado anterior puede evitarse agrupando ambas sentencias dentro de una transacción.
- Los pasos para crear una transacción en MySQL son los siguientes:
  - Iniciar una transacción con la sentencia **START TRANSACTION**.
  - Actualizar, insertar o eliminar registros en la base de datos.
  - Si se quiere que los cambios realizados en la BD sean permanentes, completar la transacción con la sentencia **COMMIT**.
  - Si sucede algún problema, podemos hacer uso de la sentencia **ROLLBACK** para cancelar los cambios realizados por las consultas que han sido ejecutadas hasta ese momento.

Ejercicio 407:

Con el modo AUTOCOMMIT activado ¿Cuál es el resultado de ejecutar los siguientes scripts?

a)
```
USE compraventa;
INSERT INTO clientes(idcliente) VALUES('a');
INSERT INTO clientes(idcliente) VALUES('b');
ROLLBACK;
INSERT INTO clientes(idcliente) VALUES('c');
```

b)
```
USE compraventa;
INSERT INTO clientes(idcliente) VALUES('a');
START TRANSACTION;
INSERT INTO clientes(idcliente) VALUES('b');
INSERT INTO clientes(idcliente) VALUES('c');
```

c) USE compraventa;

```
INSERT INTO clientes(idcliente) VALUES('a');
START TRANSACTION;
INSERT INTO clientes(idcliente) VALUES('b');
COMMIT;
INSERT INTO clientes(idcliente) VALUES('c');
```

d)
```
USE compraventa;
INSERT INTO clientes(idcliente) VALUES('a');
START TRANSACTION;
INSERT INTO clientes(idcliente) VALUES('b');
ROLLBACK;
INSERT INTO clientes(idcliente) VALUES('c');
```

5.- Índices

- Un fichero índice es un fichero asociado a una tabla o vista que acelera la búsqueda de datos.
- Cada registro de un índice está compuesto por dos campos:
    - Una **clave** de búsqueda: Contiene el valor de una o varias columnas por cada fila de una tabla.
    - Un **puntero**: Contiene la dirección de la fila en el dispositivo de almacenamiento.
- Aunque esta es la manera simple de explicar los índices, realmente es un poco más complejo, ya que MySQL almacena todas las claves del índice en una estructura de datos en árbol.
    - Esta estructura en árbol le permite a MySQL encontrar claves muy rápidamente.
- Cuando MySQL encuentra que hay un índice para una columna de búsqueda lo utiliza en vez de hacer un escaneo completo de toda la tabla.
    - Esto reduce de manera importante los tiempos de CPU y las operaciones de entrada/salida en disco.
    - También mejora la concurrencia porque MySQL bloqueará la tabla únicamente para obtener las filas que necesite en base a lo que encontró en el índice.

- Cuando manejamos tablas con grandes cantidades de datos, la mejora en la obtención de los datos mediante búsquedas indexadas puede ser muy significativa.

- No obstante lo anterior, el uso de índices tiene los siguientes inconvenientes:
  - Los índices ocupan espacio, si tenemos creados varios índices para una misma tabla, puede que el conjunto de éstos ocupe en algunos casos más que la tabla a la que hacen referencia.
  - Los índices han de actualizarse cada vez que se modifique el contenido de la columna o columnas indexadas, lo que ralentiza la actualización de datos.
- Habrá pues que estudiar en cada caso la conveniencia o no de crear un índice concreto.

5.1.- Creación de índices

Manual: *13.1.4. Sintaxis de CREATE INDEX*

- Podemos distinguir entre dos tipos de índices: de clave primaria y ordinarios.
- Los índices de clave primaria son creados por MySQL en el momento de crear una tabla con clave primaria.
- Los índices ordinarios podemos crearlos con la sentencia:

```
CREATE [ UNIQUE ]
INDEX nombre_índice
ON nombre_tabla ( {columna [ASC | DESC ] } [, ...] )
```

  - Esta sentencia crea un índice asociado a la tabla especificada.
  - La opción UNIQUE evita que se puedan repetir los valores de los campos involucrados.

        - Sólo se puede utilizar con columnas que hayan sido declaradas UNIQUE.
    - Las opciones ASC y DESC determinan que los registros del índice se ordenen en ascendente o descendente por el valor de la columna o columnas que lo componen. Por defecto se ordenan en ascendente.
        - Cada opción afecta únicamente a la columna que le precede.
- Podemos modificar o borrar un índice con las sentencias **ALTER INDEX** y **DROP INDEX**.

Ejercicio 408.

a) Crear una tabla con tres campos, los dos primeros formarán la clave primaria. Indexarla por el tercer campo. Comprobar los índices que se han creado.

b) Crear dos índices sobre la tabla empleados, el primero se llamará "apenom" y deberá estar ordenado en ascendente por apellido y nombre, el segundo se llamará "fechanaci" y estará ordenado en descendente por fecha de nacimiento.

- El gestor de BBDD posee un "optimizador de consultas" que decide cuál de los índices creados es el más adecuado para realizar las búsquedas en las tablas.
- No obstante, si queremos que el motor realice las búsquedas sólo a partir de determinados índices, al final de la cláusula FOR de una sentencia SELECT escribiremos:

use index (lista-de-índices)

- Análogamente, si no queremos que el optimizador utilice determinados índices en las búsquedas escribiremos:

ignore index (lista-de-índices)

Ejercicio 409.

a) Crear 2 índices sobre la tabla PRODUCTOS, uno con la columna IdProveedor y otro con la columna IdCategoría.

b) Sentencia que visualice el contenido de aquellas filas de la tabla PRODUCTOS que no son provistos por el proveedor "Exotic Liquids" realizando una búsqueda indexada por IdProveedor.

c) Sentencia que visualice el contenido de aquellas filas de la tabla PRODUCTOS pertenecientes a la categoría "Carnes" sin realizar una búsqueda indexada por IdCategoría.

d) Borrar los índices creados en el apartado a)

6.- Vistas.

Manual: *21. Vistas.*

- Una vista es una tabla temporal cuyo contenido está definido por una consulta.
    - El script que crea la consulta es almacenado por el gestor en la BD.
- Las vistas comparten espacio de nombres con las BD, por lo que no pueden denominarse igual que ellas.
- Una vista puede utilizarse en una instrucción SQL de modo similar a como se utiliza una tabla base.
    - Cuando, durante la ejecución de una instrucción SQL, el gestor encuentra el nombre de una vista, ejecuta el script que tiene almacenado con dicho nombre y crea una tabla temporal con el resultado de dicha ejecución.
- Dentro de una vista puede hacerse referencia a una o varias tablas, o bien a otras vistas.
- Las vistas suelen utilizarse para centrar, simplificar y personalizar la percepción de la BD para cada usuario.
- La utilización de vistas permite que los distintos usuarios y/o programas de aplicación tengan acceso únicamente a los datos contenidos en ellas, pero no al resto de datos contenidos en las tablas base subyacentes.

6.1.- Sintaxis de creación de vistas:

CREATE [OR REPLACE] VIEW *nombre_vista* [(*columnas*)]
AS *sentencia_select*

- nombre_vista: ha de ser único en la BD.
  - En principio, las vistas se crean en la BD abierta. Si queremos que se cree en una BD distinta debemos calificar el nombre de la vista con el nombre de la BD
- columnas: Lista de los identificadores de columna de la vista.
  - Esta opción se utiliza cuando queremos que los nombres de columnas de nuestra vista sean diferentes de los obtenidos como resultado de la consulta que contiene.
  - Si no se especifica, las columnas de la vista adquieren el identificador especificado en la lista de selección de SELECT.
- AS: Indica las acciones que a llevar a cabo cuando se ejecute la vista.
  - Estas acciones tendrán la forma de una sentencia SELECT, pudiendo ser ésta de cualquier complejidad, con ciertas restricciones.

- Cuando se hace referencia a una vista dentro de una instrucción SQL, el sistema comprueba que todos los objetos a los que se hace referencia en el script de creación de la vista existen, que son válidos en el contexto de la instrucción y, en caso de que la consulta implique modificación de datos, que dicha modificación no infringe ninguna regla de integridad.
  - Las comprobaciones correctas convierten la acción de la instrucción SQL en una acción contra las tablas subyacentes, las no correctas devuelven un mensaje de error.
- Podemos modificar o borrar una vista con las sentencias **ALTER VIEW** y **DROP VIEW**.

- También posemos mostrar el contenido de una vista mediante la sentencia **SHOW CREATE VIEW**

Ejercicio 410.

Crear una vista que contenga los nombres y precios de todos los productos de la categoría 'Condimentos'. Los nombres de las columnas deberán ser "Nombre" y "Precio".
Ejecutarla.
Visualizar su contenido.

a) Utilizando inner join.

b) Utilizando una subconsulta.

Ejercicio 411.

Crear las siguientes vistas en la BD COMPRAVENTA.
Cada ejercicio debéis realizarlo de 2 formas: utilizando inner join y sin utilizarlo.

a) Vista para obtener el nombre completo de los empleados que tienen asignada la región norte.

b) Vista para obtener el nombre completo del empleado que tiene asignados más territorios.

c) Vista para obtener el nombre completo de los empleados que son jefes.

d) Idem de los empleados que no son jefes.

e) Vista para obtener el nombre de la categoría que tiene más productos.

f) Vista para obtener el nombre del cliente que hizo el pedido de mayor valor en marzo de 1998.

g) Vista para obtener el identificador de cada pedido y su valor total, incluyendo descuentos y cargos.