

BASES DE DATOS

Desarrollo de Aplicaciones Multiplataforma

Desarrollo de Aplicaciones Web

GESTIÓN DE BASES DE DATOS

Administración de Sistemas Informáticos en Red

PROGRAMACIÓN DE BASES DE DATOS en MySQL

Luis Dorado

Vanesa Martínez

Pablo Bahillo

Alba Tortosa



Ejercicios

Contenido

0	CREACIÓN DE BDS, TABLAS E INSERCIÓN DE DATOS	3
0.1	Creación de BDs y tablas	3
0.1.1	BD 'bdLiga'	3
0.1.2	BD 'bdFormacion'	3
0.1.3	BD 'bdEmpresa'	4
0.1.1	BD 'bdTiendas'	4
0.1.2	BD 'bdTrabajadoresEdificios'	5
0.1.3	BD 'bdEmpleadosOficinas'	5
0.1.4	BD 'bdUniversidad'	6
0.1.5	BD 'bdPedidos'	6
0.1.6	BD 'bdJardineria'	7
0.1.1	BD 'bdCirco'	7
0.2	Insertar datos en las bases de datos	8
1	PROGRAMACIÓN DE BBDD EN MYSQL I	9
1.1	Procedimientos sin parámetros	9
1.1.1	Sin base de datos	9
1.1.2	BD Liga	9
1.2	Procedimientos con parámetros	10
1.2.1	BD Jardinería	10
1.2.2	BD Pedidos	11
1.2.3	Sin base de datos	11
1.2.4	BD Liga	12
1.3	Variables	13
1.3.1	BD Liga	13
1.3.2	Sin base de datos	13
1.3.3	BD Jardinería	14
2	PROGRAMACIÓN DE BBDD EN MYSQL II	15
2.1	Estructuras de control condicionales simples	15
2.1.1	Estructuras IF . . . ELSE (sin base de datos)	15
2.1.2	Estructuras IF . . . ELSE (BD Pedidos)	15
2.1.3	Estructuras IF . . . ELSEIF . . . ELSE (sin base de datos)–	15
2.1.4	Estructuras CASE (sin base de datos)	16
2.2	Estructuras de control condicionales complejas y/o anidadas	17
2.2.1	Sin base de datos	17
2.2.2	BD Formación	17
2.2.3	BD Empresa	18
2.2.4	BD Universidad	20
2.3	Estructuras de control iterativas no anidadas	22
2.3.1	Sin base de datos	22
2.4	Estructuras de control iterativas anidadas	26
2.4.1	Sin base de datos	27
2.5	Funciones	28
2.5.1	BD Empresa	29
2.5.2	BD Jardinería	30
2.5.3	Sin base de datos	31
2.6	Cursores	36
2.6.1	BD Empresa	36
2.6.2	BD Pedidos	37
2.6.3	BD Tiendas	38
2.6.4	BD Circo	38

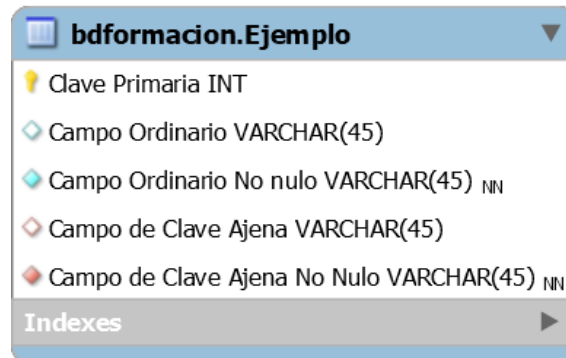
2.6.5	BD Formación	40
3	PROGRAMACIÓN DE BBDD EN MYSQL III	43
3.1	Control de errores.....	43
3.1.1	Sin base de datos (Captura de excepciones de sistema).....	43
3.1.2	BD Universidad (Captura de excepciones de sistema).....	44
3.1.3	BD Circo (Captura de excepciones de sistema y de usuario).....	46
3.2	Transacciones I	50
3.2.1	Sin base de datos	50
3.2.2	BD Circo.....	53
3.2.3	BD Formación	55
3.3	Transacciones II	59
3.3.1	Sin base de datos	59
3.4	Triggers.....	59
3.4.1	BD Formación	59
3.4.2	BD Circo.....	60
3.4.3	BD Empresa (Falta solución)	66

0 CREACIÓN DE BDS, TABLAS E INSERCIÓN DE DATOS

Crea las bases de datos que se te vayan indicando, sus tablas e inserta las filas en sus tablas.

Notas:

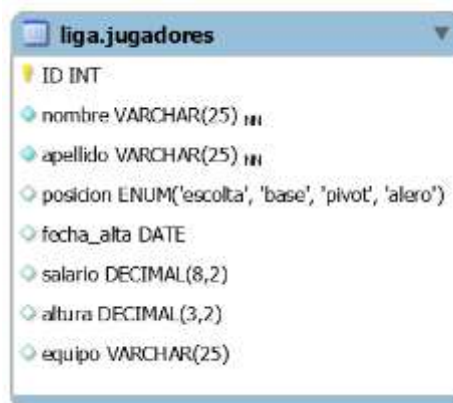
- ✓ Comprueba que existan las BDs y las tablas antes de crearlas.
- ✓ Créalas cuando las vayas necesitando para resolver los ejercicios, no es necesario que las crees todas al inicio.
- ✓ **Recuerda: ¿Qué significan los colores asignados a cada campo?**



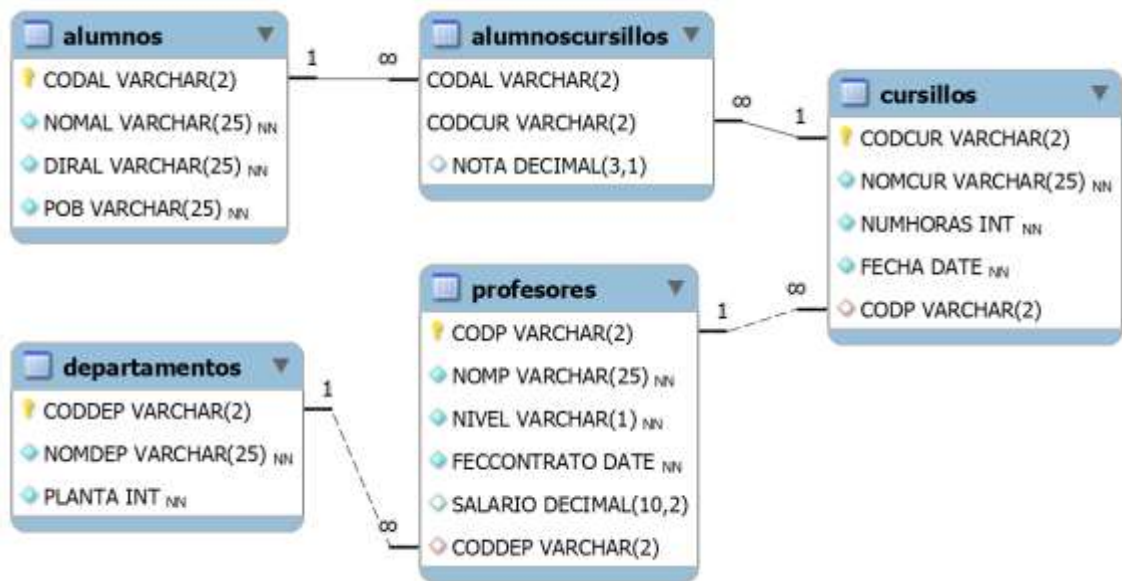
- Clave primaria (puede pertenecer a dos campos si es compuesta).
- Campo ordinario.
- Campo ordinario no nulo.
- Campo de clave ajena.
- Campo de clave ajena no nula.

0.1 Creación de BDs y tablas

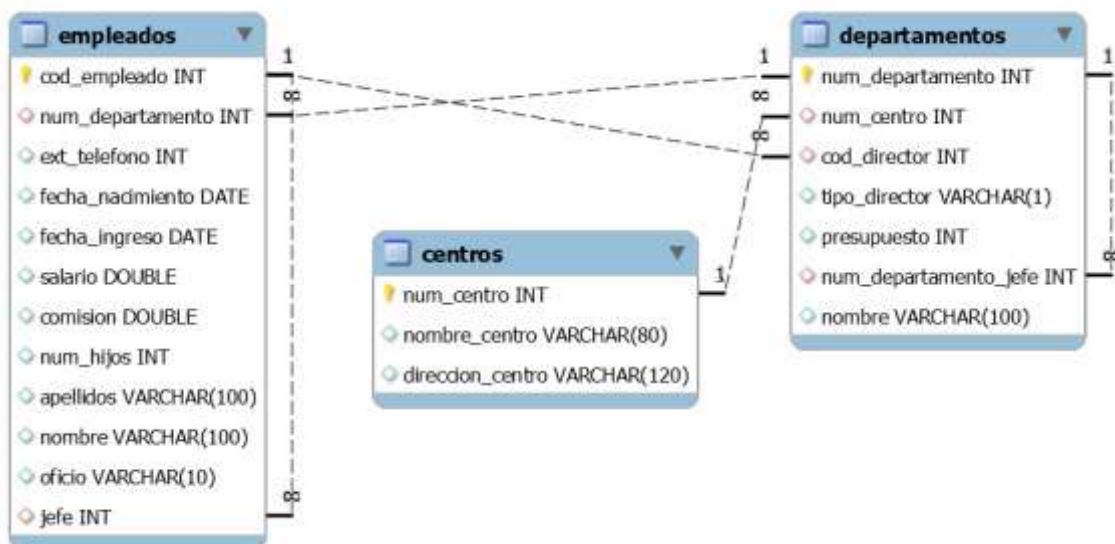
0.1.1 BD 'bdLiga'



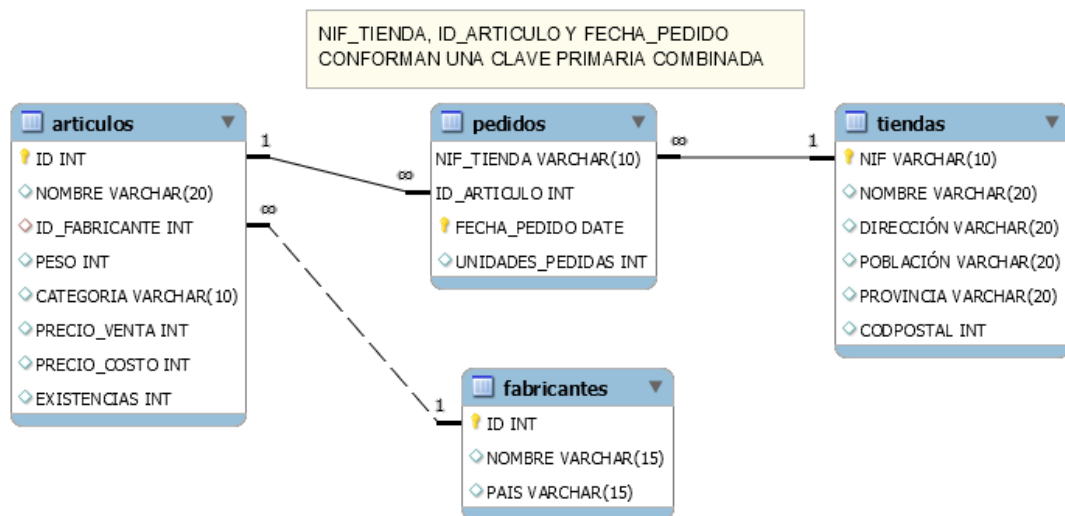
0.1.2 BD 'bdFormacion'



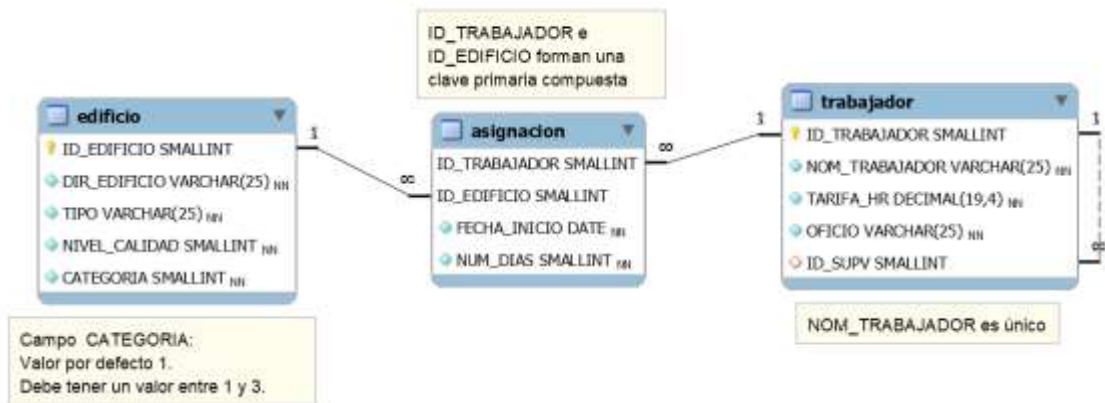
0.1.3 BD 'bdEmpresa'



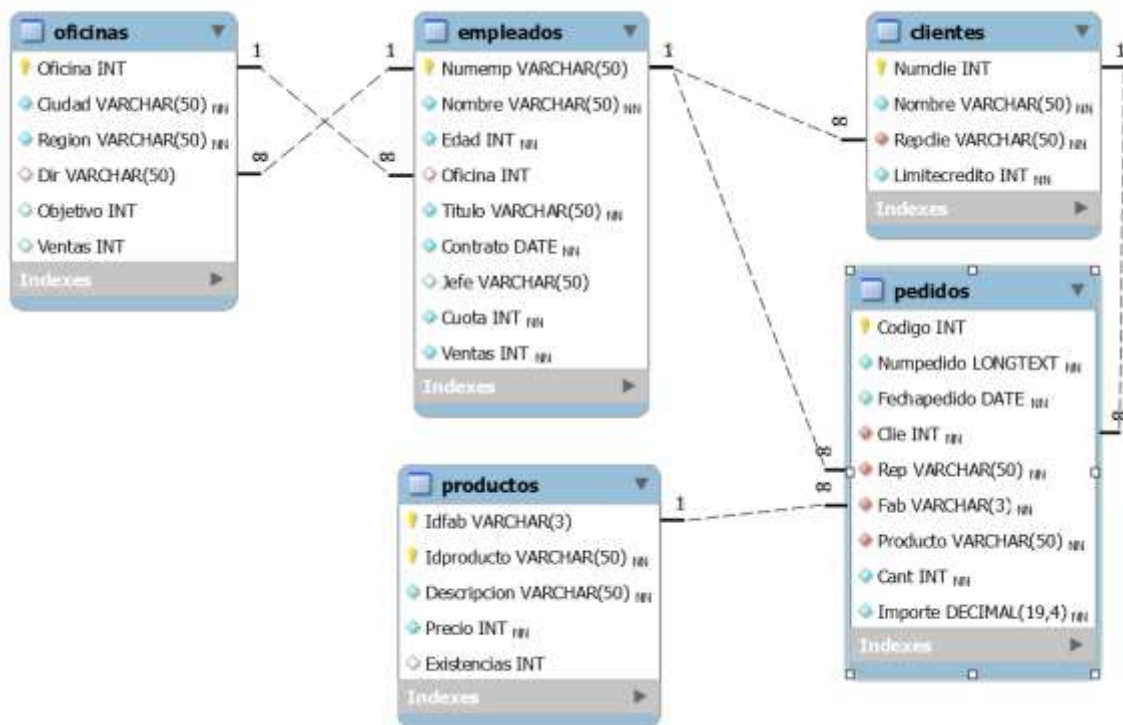
0.1.1 BD 'bdTiendas'



0.1.2 BD 'bdTrabajadoresEdificios'

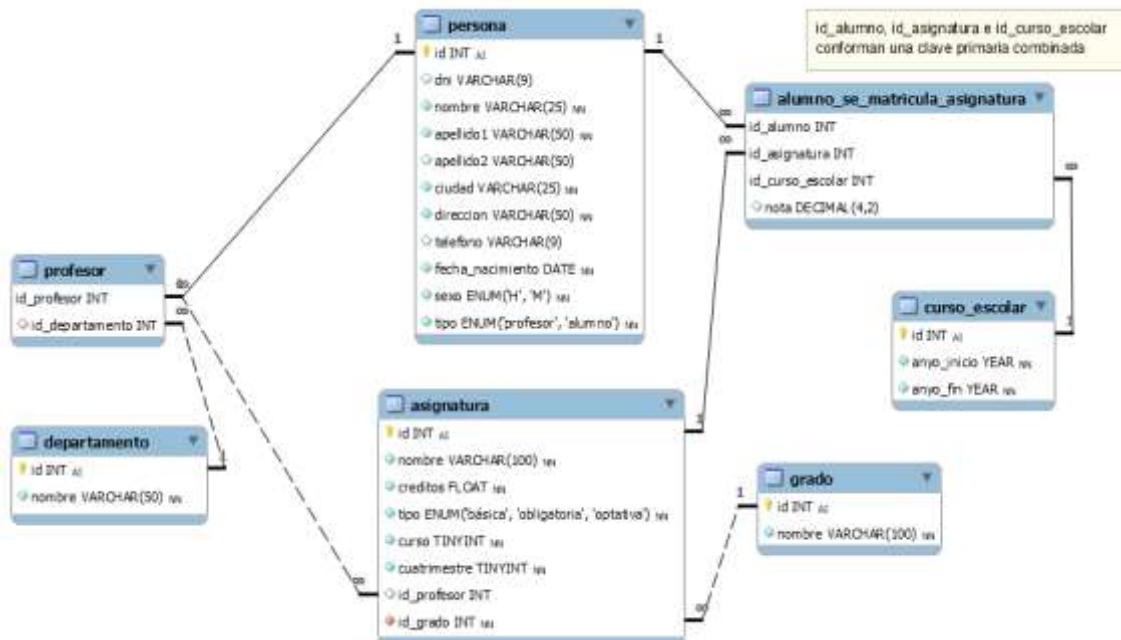


0.1.3 BD 'bdEmpleadosOficinas'

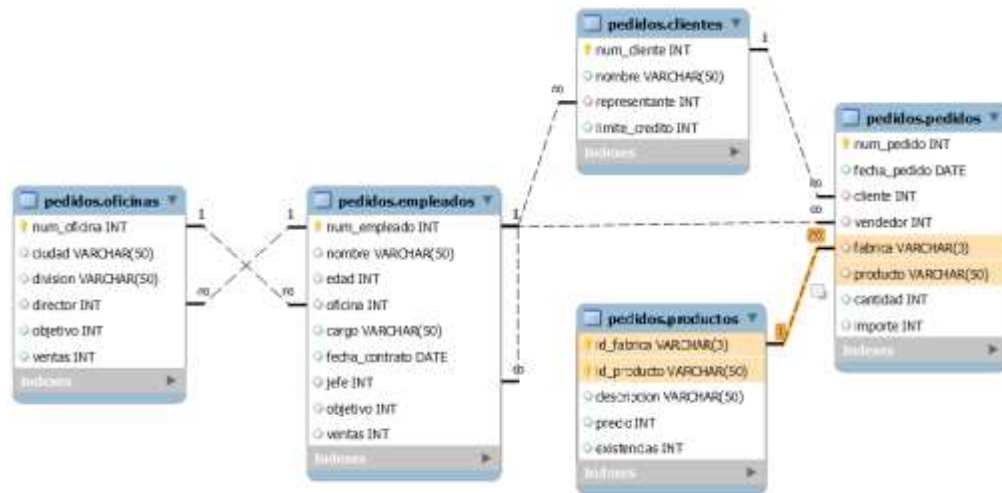


Nota importante: Como oficinas y empleados se hallan relacionados doblemente en ambas direcciones deberán dejar al menos estas tablas sin restricción de clave ajena hasta que hayas realizado las inserciones.

0.1.4 BD 'bdUniversidad'

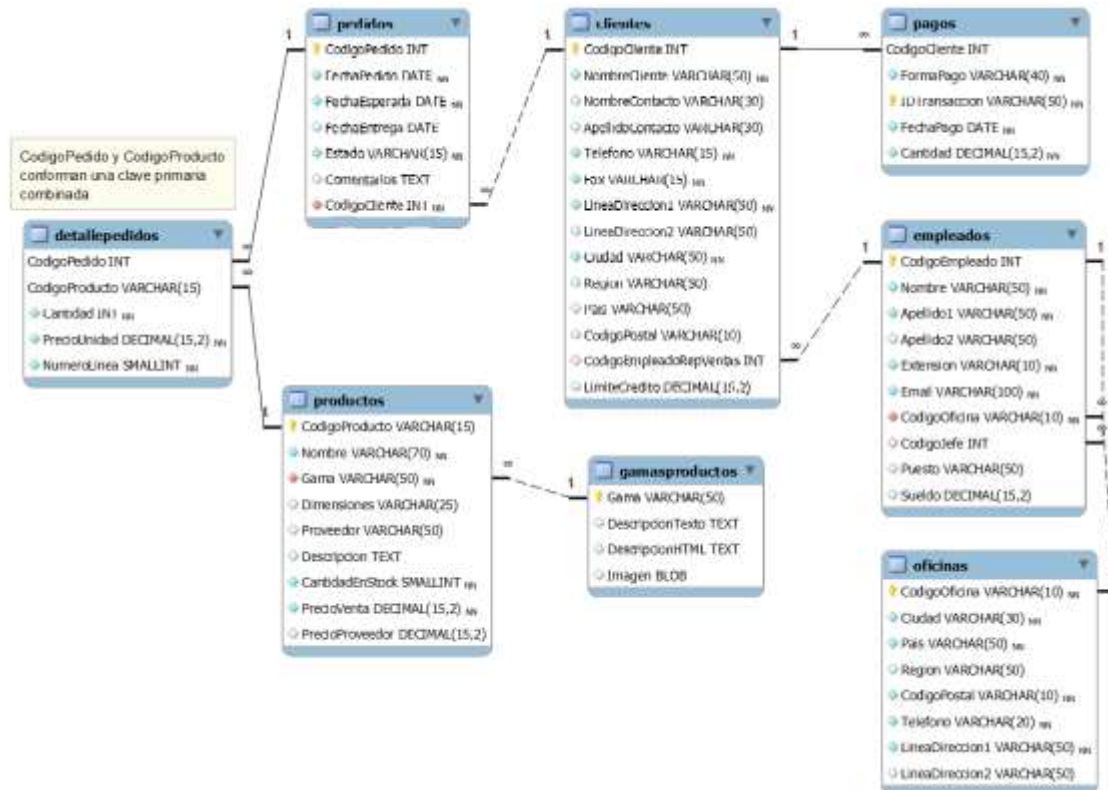


0.1.5 BD 'bdPedidos'

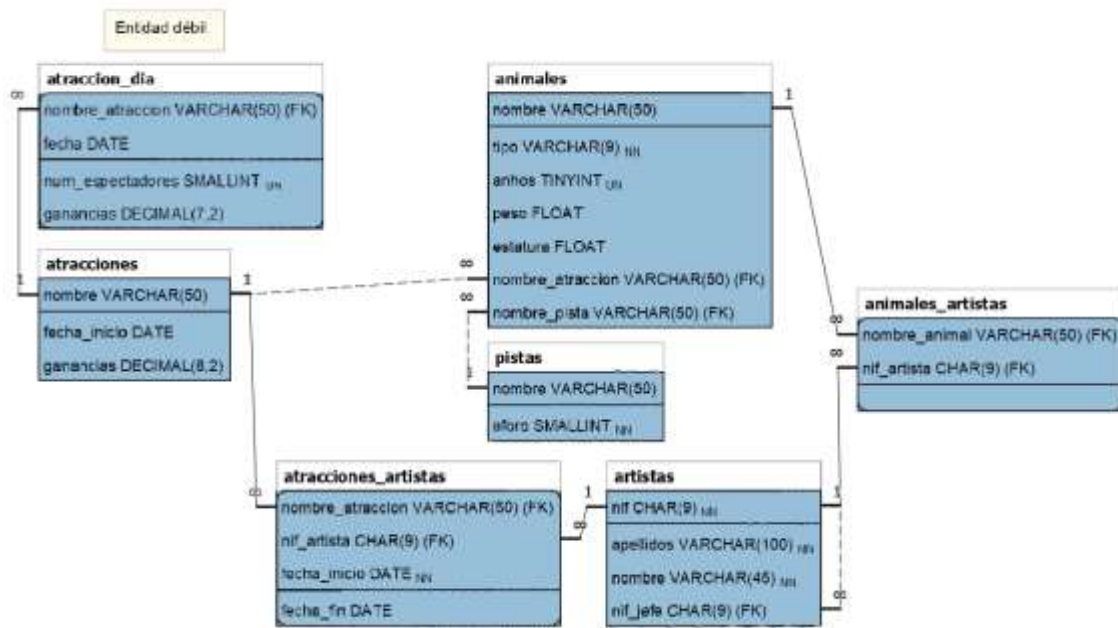


- ✓ Como la clave primaria de *productos* es compuesta, también debe serlo la clave ajena de *pedidos* que apunta a *productos*.

0.1.6 BD 'bdJardineria'



0.1.1 BD 'bdCirco'



0.2 Insertar datos en las bases de datos

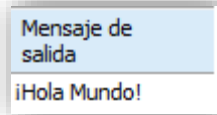
Las inserciones ya se habrán realizado o bien se hallarán en la plataforma Moodle del módulo.

1 PROGRAMACIÓN DE BBDD EN MYSQL I

1.1 Procedimientos sin parámetros

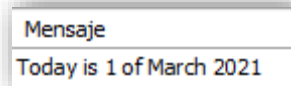
1.1.1 Sin base de datos

- 1.1.1.1** Escribe un procedimiento denominado “holaMundo” que no tenga ningún parámetro de entrada ni de salida y que muestre el texto ¡Hola mundo!



DIFICULTAD BAJA

- 1.1.1.2** Escribe un procedimiento denominado ‘muestraFecha’ que muestre la fecha actual con un mensaje de este tipo.



DIFICULTAD MEDIA

1.1.2 BD Liga

- 1.1.2.1** Escribe un procedimiento denominado ‘listaEquipos’ que muestre la lista de equipos y el número de jugadores que hay en cada uno.

equipo	Nº de Jugadores
Regal Barcelona	2
Real Madrid	3
Caja Laboral	2
CAI Zaragoza	3

DIFICULTAD BAJA

1.2 Procedimientos con parámetros

1.2.1 BD Jardinería

- 1.2.1.1** Escribe un procedimiento denominado "clientesPais" que reciba el nombre de un país como parámetro de entrada y realice una consulta sobre la tabla `cliente` para obtener todos los clientes (nombre, ciudad y país) que existen en la tabla de ese país y los muestre por pantalla.

18 • `CALL clientesPais('Australia');`

Result Grid | Filter Rows: | Export

NombreCliente	Ciudad	Pais
Tutifrutí S.A	Sydney	Australia
El Jardín Viviente S.L	Sydney	Australia

DIFICULTAD BAJA

- 1.2.1.2** Escribe un procedimiento denominado "pagoMaximo" que reciba como parámetro de entrada una forma de pago, que será una cadena de caracteres (Ejemplo: `PayPal`, `Transferencia`, etc). Y que muestre por pantalla el pago de máximo valor realizado para esa forma de pago.

32 • `CALL pagoMaximo('PayPal');`

Result Grid | Filter Rows: | Export:

Mensaje
El pago máximo que se ha hecho con PayPal es 20000.00

DIFICULTAD BAJA

- 1.2.1.3** Escribe un procedimiento denominado "infoPagos" que reciba como parámetro de entrada una forma de pago, que será una cadena de caracteres (Ejemplo: `PayPal`, `Transferencia`, etc.) y que muestre por pantalla los siguientes valores teniendo en cuenta la forma de pago seleccionada como parámetro de entrada:

- ✓ el pago de máximo valor.
- ✓ el pago de mínimo valor.
- ✓ el valor medio de los pagos realizados.
- ✓ la suma de todos los pagos, el número de pagos realizados para esa forma de pago.

1 • `CALL infoPagos('PayPal');`

Result Grid | Filter Rows: | Export: | Write

Pago máximo	Pago Mínimo	Pago medio	Pagos totales
20000.00	232.00	7156.500000	171756.00

DIFICULTAD BAJA

1.2.2 BD Pedidos

1.2.2.1 Escribe un procedimiento denominado 'cambiarNombreCliente' que actualice el nombre de un cliente.

- ✓ Ayuda: recibe dos parámetros, numero de cliente de quien se va actualizar y el nuevo nombre.

```
CALL cambiarNombreCliente(2101, 'Pepe Sanz');
```

DIFICULTAD BAJA

1.2.2.2 Escribe un procedimiento denominado 'borrarPedido' Que elimine un pedido.

- ✓ Ayuda: recibe un parámetro, numero de pedido que se va eliminar

```
CALL borrarPedido(110035);
```

DIFICULTAD BAJA

1.2.3 Sin base de datos

1.2.3.1 Escribe un procedimiento denominado 'holaNombre' que reciba un nombre (por ejemplo, Alba) como parámetro y muestre el saludo de tipo *¡Hola Alba!*

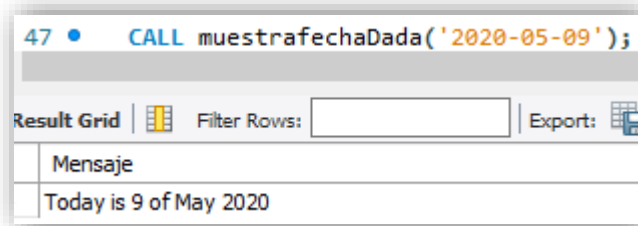
31 • `CALL holaNombre('Luis');`

Result Grid | Filter Rows: | Export: | Write

Mensaje de saludo
¡Hola Luis!

DIFICULTAD BAJA

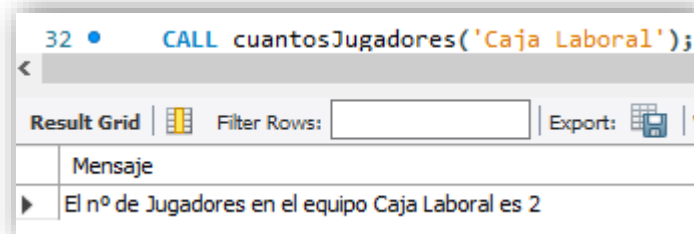
- 1.2.3.2** Escribe un procedimiento denominado 'muestraFechaDada' que reciba una fecha (por ejemplo, '2020-05-09') como parámetro y muestre un mensaje de este tipo.



DIFICULTAD MEDIA

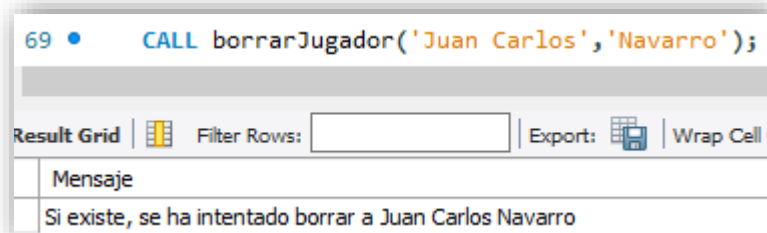
1.2.4 BD Liga

- 1.2.4.1** Escribe un procedimiento denominado "cuantosJugadores" que, dado un nombre de equipo, muestre el número de jugadores de ese equipo.



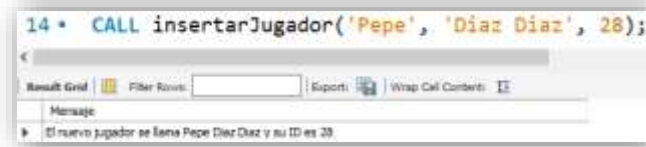
DIFICULTAD BAJA

- 1.2.4.2** Escribe un procedimiento denominado "borrarJugador" que, dado un nombre y un apellido, borre dicho jugador.



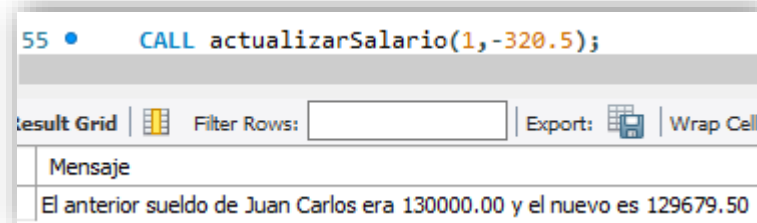
DIFICULTAD BAJA

- 1.2.4.3** Escribe un procedimiento denominado "insertarJugador" que permita crear un nuevo jugador e informe por pantalla de su creación con todos sus datos. Deberá usar como parámetros ID, nombre y apellidos (del mismo tipo que los campos de la tabla). El resto de campos del jugador serán nulos.



DIFICULTAD MEDIA

- 1.2.4.4** Escribe un procedimiento denominado "actualizarSueldo" que, dado un id de jugador y un numero (positivo o negativo) actualice el sueldo de dicho jugador sumando/restando la cantidad e informe del resultado por pantalla. Pista: Primero el UPDATE y luego el SELECT.



DIFICULTAD MEDIA

1.3 Variables

1.3.1 BD Liga

- 1.3.1.1** Reescribe el procedimiento "cuantosJugadores" usando dos variables locales: en una almacenarás el número de jugadores solicitado (con **SET**) y en la otra almacenarás el mensaje de salida (con **SET**). Después mostrarás el valor de la variable "mensaje".

DIFICULTAD BAJA

1.3.2 Sin base de datos

- 1.3.2.1** Reescribe el procedimiento "holaNombre" usando una variable local donde almacenarás el mensaje de salida para después mostrar el valor de esta variable. Pista: Usa **SET** y **CONCAT()**.

DIFICULTAD BAJA

- 1.3.2.2** Reescribe el procedimiento *“muestraFechaDada”* usando 4 variables locales: en 3 almacenarás, respectivamente, el día, mes y año de la fecha proporcionada y en la otra almacenarás el mensaje de salida. Después mostrarás el valor de la variable *“mensaje”*. Pista: Usa **SET** y **CONCAT()**.

DIFICULTAD MEDIA

1.3.3 BD Jardinería

- 1.3.3.1** Reescribe el procedimiento *infoPagos* usando 5 variables locales: en 4 almacenarás, respectivamente, el máximo, mínimo, media y total de los pagos (con **SELECT INTO**) y en la otra almacenarás el mensaje de salida (con **SET**). Después mostrarás el valor de la variable *“mensaje”*.

DIFICULTAD MEDIA

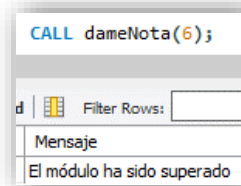
2 PROGRAMACIÓN DE BBDD EN MYSQL II

2.1 Estructuras de control condicionales simples

2.1.1 Estructuras IF ... ELSE (sin base de datos)

- 2.1.1.1** Escribe un procedimiento “dameNota” que reciba un número real de entrada, que represente el valor de la nota de un alumno, y muestre un mensaje indicando si ha superado o no el módulo:

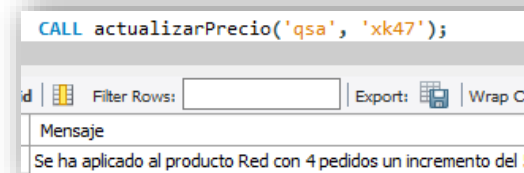
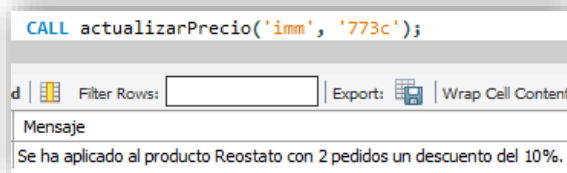
- ✓ [0,5) = No superado
- ✓ [5,10] = Superado



DIFICULTAD BAJA

2.1.2 Estructuras IF ... ELSE (BD Pedidos)

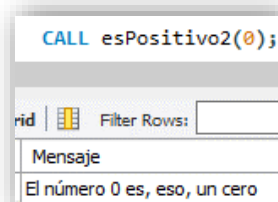
- 2.1.2.1** Escribe un procedimiento denominado ‘actualizarPrecio’ que reciba una fábrica y un producto como parámetro de entrada. Si el producto se ha vendido más de 3 veces, aumentarle el precio un 5%. Si no, rebajarle el precio un 10%.



DIFICULTAD MEDIA

2.1.3 Estructuras IF ... ELSEIF ... ELSE (sin base de datos)

- 2.1.3.1** Escribe un procedimiento llamado “esPositivo” que reciba un número real de entrada y muestre un mensaje indicando si el número es positivo, negativo o cero. Haz uso de la estructura de control IF.

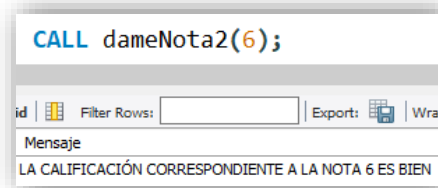


DIFICULTAD BAJA

2.1.3.2 Escribe un procedimiento “dameNota2”

que reciba un número real de entrada, que represente el valor de la nota de un alumno, y muestre un mensaje indicando qué nota ha obtenido teniendo en cuenta las siguientes condiciones:

- ✓ $[0,5)$ = Insuficiente
- ✓ $[5,6)$ = Aprobado
- ✓ $[6, 7)$ = Bien
- ✓ $[7, 9)$ = Notable
- ✓ $[9, 10]$ = Sobresaliente
- ✓ En cualquier otro caso la nota no será válida.

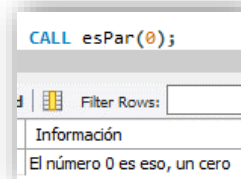
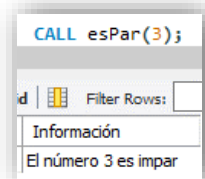


DIFICULTAD BAJA

2.1.3.3 Escribe un procedimiento “esPar” que te muestre por pantalla si un número es par, impar o 0.

Notas:

- ✓ Un número es par si al dividirlo entre dos da como resto 0.
- ✓ Si da como resto uno (el otro caso posible) será impar.
- ✓ Para saber el resto de una división entera se usa el operador %.
- ✓ Si $\text{num} \% 2 = 0$ entonces num es par. Si $\text{num} \% 2 = 1$ entonces num es impar.



DIFICULTAD MEDIA

2.1.4 Estructuras CASE (sin base de datos)**2.1.4.1** Modifica el procedimiento

“esPositivo2” haciendo uso de la estructura de control CASE. Llámalo “esPositivoCase”.

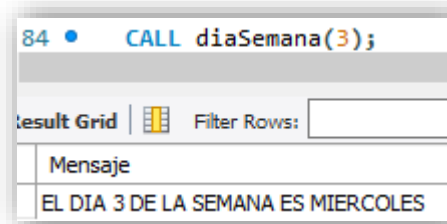
DIFICULTAD BAJA

- 2.1.4.2** Modifica el procedimiento “dameNota2” haciéndolo de la estructura de control **CASE**. Llámalo “dameNotaCase”.

DIFICULTAD BAJA

- 2.1.4.3** Escribe un procedimiento llamado “diaSemana” que reciba como parámetro de entrada un valor numérico que represente un día de la semana y que escriba por pantalla una cadena de caracteres con el nombre del día de la semana correspondiente. Usa **CASE**.

- ✓ Por ejemplo, para el valor de entrada **1** debería devolver la cadena ‘**El día de la semana es lunes**’.

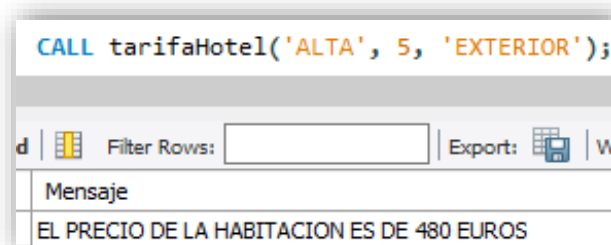


DIFICULTAD BAJA

2.2 Estructuras de control condicionales complejas y/o anidadas

2.2.1 Sin base de datos

- 2.2.1.1** Implementar un script llamado ‘tarifaHotel’ que calcule (y muestre) el precio de una habitación de un hotel en base a la temporada (ALTA, MEDIA o BAJA), el número de noches y el tipo de habitación (EXTERIOR o INTERIOR). El procedimiento usará 3 parámetros: la temporada, el nº de noches y el tipo de habitación. El precio BASE por noche de una habitación EXTERIOR es de 80 € y el de la interior es de 60 €. En temporada ALTA se aplica un incremento del 20% al precio base y en la BAJA un descuento del 10%. Se mostrará la información en la pantalla de la siguiente manera:

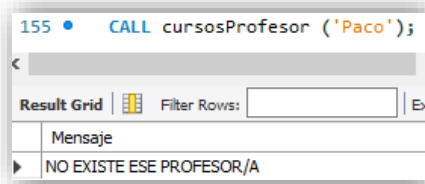
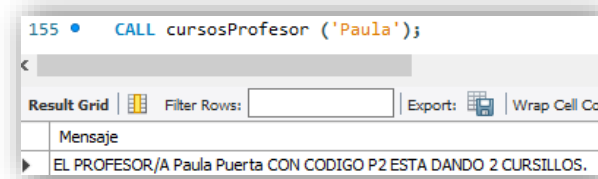
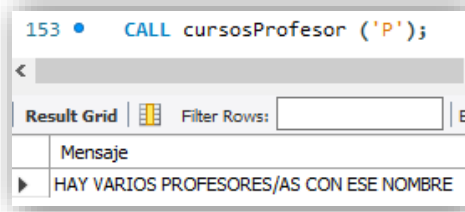


DIFICULTAD MEDIA

2.2.2 BD Formación

2.2.2.1 Implementar un script llamado `cursosProfesor` en la base de datos bdFormacion que, dado un nombre de profesor (cuyo valor recibiremos como parámetro), muestre su código de profesor y la cantidad de cursos que ha impartido.

- ✓ El nombre del profesor tan solo debe contener el valor parámetro, no tiene porqué ser idéntico.
- ✓ Si el profesor no existe o hay varios profesores con el mismo nombre se mostrarán mensajes diferentes.
- ✓ El mensaje relativo a los cursos impartidos debe ser distinto si no ha impartido ningún curso, si sólo ha impartido uno o si han sido varios los cursos impartidos.
- ✓ Truqui: `campotabla LIKE CONCAT('%', nombreParámetro, '%')`;



Etc...

DIFICULTAD MEDIA

2.2.3 BD Empresa

2.2.3.1 Crea un procedimiento llamado `insertarEmpleado` para insertar empleados. Recibirá como parámetro nombre, apellidos, oficio y departamento. Además de insertar los campos recibidos por parámetro también fijarás el salario y la comisión usando los criterios que se exponen a continuación.

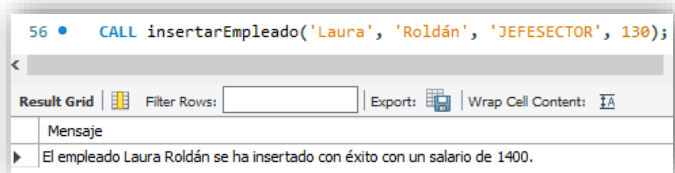
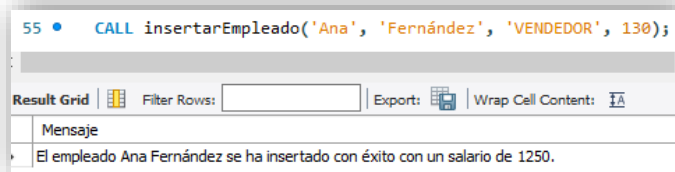
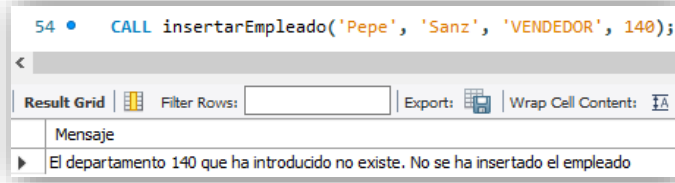
- ✓ Según el oficio asigne un salario fijo: EMPLEADO (1200€), VENDEDOR (1250€), ANALISTA (1300€), JEFESECTOR (1400€)
- ✓ Asígnale a todos una comisión fija de 0€.

Una vez creado, úsalo para insertar varios empleados con diferentes oficios.

Notas:

- ✓ Para evitar errores por infracción de clave ajena, debes comprobar si existe el departamento y, en ese caso, mostrar **un** mensaje. En caso de que se realice correctamente la inserción, informa de ello por pantalla.
- ✓ Usa la construcción **IF EXISTS (SELECT * FROM ...) THEN** para comprobar si existe el departamento.
- ✓ Usa solo IFs.

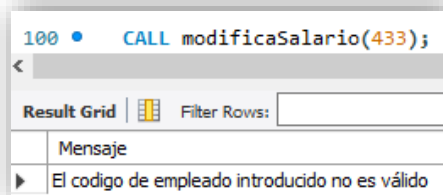
- ✓ **Pista:** Para escoger el valor de la clave primaria a insertar puedes consultar el valor máximo de todos los valores de la clave primaria de la tabla para luego insertar ese valor incrementado en una unidad.

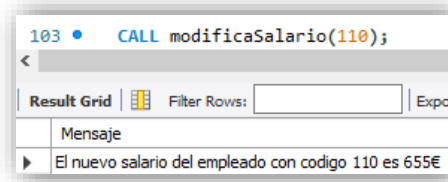


DIFICULTAD MEDIA

2.2.3.2 Crea un procedimiento llamado “modificaSalario” que reciba un código de empleado y modifique su salario.

- ✓ Si el empleado tiene oficio "PRESIDENTE", baja un 50% su salario.
- ✓ Si el empleado tiene oficio "JEFESECTOR", baja un 10% su salario.
- ✓ Si tiene oficio "VENDEDOR" sube un 10% su salario.
- ✓ Si el empleado tiene oficio "EMPLEADO" aumenta un 20% su salario.
- ✓ Si el código de empleado no existe muestra un error y sino muestra el nuevo salario.
- ✓ Cualquier otro dejará el sueldo igual.
- Usa la construcción **CASE** y la estructura **IF**.
- Usa la construcción **IF paramCodEmp NOT IN (SELECT ...)** **THEN** para comprobar si existe el departamento.

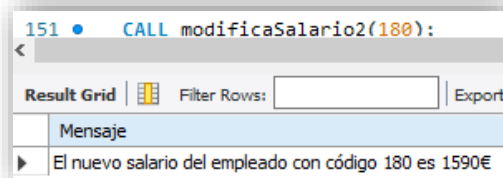
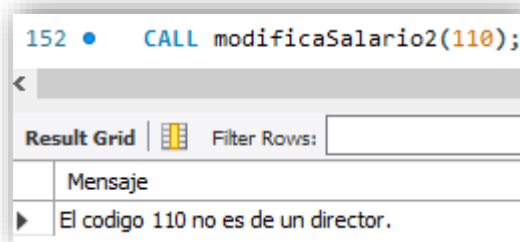




DIFICULTAD MEDIA

2.2.3.3 Crea un procedimiento llamado “modificaSalario2” para modificar el salario de un empleado especificado en función del Nº de empleados (en total) de todos los departamentos que dirige:

- ✓ Si no dirige ningún departamento se mostrará un error.
- ✓ Si los departamentos que dirige no tienen ningún empleado -> la subida será 50€
- ✓ Si tienen un empleado -> la subida será 80€
- ✓ Si tienen 2 empleados -> la subida será de 100€
- ✓ Si tienen 3 o más -> la subida será 110€
- ✓ Usa la construcción **CASE** y la estructura **IF**.



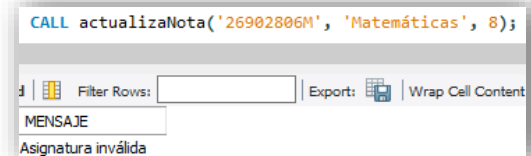
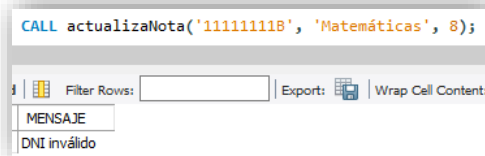
DIFICULTAD MEDIA

2.2.4 BDUiversidad

2.2.4.1 Crea un procedimiento llamado “actualizaNota” que actualice la nota de un estudiante en una asignatura en el curso actual (2020/2021).

- ✓ El procedimiento debe recibir como parámetro un DNI, un nombre de asignatura y una nota
- ✓ Si el estudiante no existe, mostrará el texto 'DNI inválido'

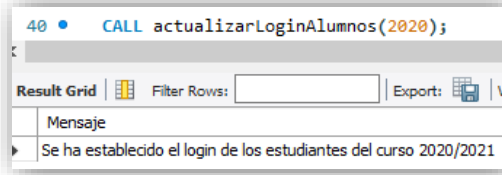
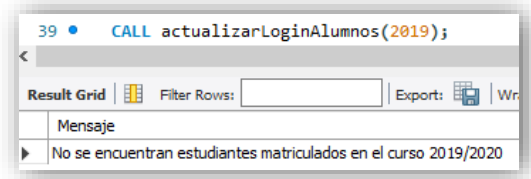
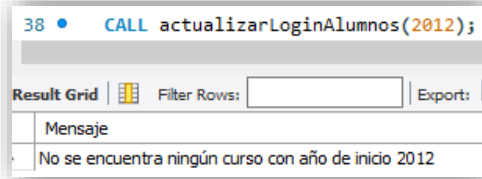
- ✓ Si la asignatura no existe, mostrará el texto 'Asignatura inválida'
- ✓ Si el estudiante no está matriculado en dicha asignatura en el curso actual, mostrará el texto 'No existe la matrícula'
- ✓ En cualquier otro caso, se actualizará la nota del estudiante indicado en la asignatura indicada en el curso actual.
- ✓ Una vez creado el procedimiento, utilízalo 4 veces, uno con cada resultado definido.



DIFICULTAD ALTA

2.2.4.2 Crea un procedimiento llamado “creaLogins” que reciba como parámetro un año y actualice el campo login de todos los estudiantes (personas de tipo ‘alumno’) matriculados en dicho año según las siguientes reglas:

- ✓ El año deberá ser el año de inicio. En el curso 2020/2021 el año de inicio sería 2020.
- ✓ Si el año no se corresponde con ningún año de inicio, mostrará el texto 'Año inválido'.
- ✓ El login estará compuesto de la primera letra del nombre, seguido de un guion bajo '_', seguido de las tres primeras letras del primer apellido, seguido de otro guion bajo, seguido de los 2 últimos caracteres del DNI, seguido de otro guion bajo, seguido de los dos últimos dígitos del año de inicio y los dos últimos dígitos del año de fin.
- ✓ Ejemplo: el estudiante 'Juan Sáez Vega' en el curso 2020/2021 tendrá como login 'j_sae_9S_1920'
- ✓ Una vez creado el procedimiento, utilízalo para actualizar el login de todos los estudiantes matriculados en el curso 2020/2021.



DIFICULTAD ALTA

2.3 Estructuras de control iterativas no anidadas

2.3.1 Sin base de datos

- 2.3.1.1** Crea un procedimiento llamado “secuenciaNumeros” que, dado un número como parámetro, muestre todo el intervalo de números desde el 1 hasta el número de la siguiente manera:
- ✓ Ejecución: CALL secuenciaNumeros (11);
 - ✓ Respuesta por pantalla: ‘Los números del intervalo desde 1 hasta 11 son 1 2 3 4 5 6 7 8 9 10 11.’
 - ✓ Ojo: Debe comprobar que el número es positivo.

DIFICULTAD BAJA

- 2.3.1.2** Crea un procedimiento llamado “secuenciaNumeros2” que, dado dos números como parámetro, muestre todo el intervalo de números desde el primer parámetro hasta el segundo parámetro de la siguiente manera:
- ✓ Ejecución: CALL secuenciaNumeros (3,9);
 - ✓ Respuesta por pantalla: ‘Los números del intervalo desde 3 hasta 12 son 3 4 5 6 7 8 9’;
 - ✓ Ojo: Debe comprobar que los números son positivos y el segundo mayor que el primero.

DIFICULTAD BAJA

- 2.3.1.3** Crea un procedimiento llamado “secuenciaNumeros3” que amplíe la funcionalidad del anterior ejercicio usando comas entre los números:
- ✓ Ejecución: CALL secuenciaNumeros (3,9)
 - ✓ Respuesta por pantalla: ‘Los números del intervalo desde 3 hasta 9 son 3, 4, 5, 6, 7, 8, 9.’
 - ¡Ojo que tras el último número no hay coma!

- Ojo: Debe comprobar que los números son positivos y el segundo mayor que el primero.

DIFICULTAD BAJA

2.3.1.4 Crea un procedimiento llamado “tablaMultiplicar” que recibirá un número entero entre 1 y 9 para que haga lo siguiente:

1. Borre si existe y después cree una tabla llamada TablaMul con 3 columnas de tipo número entero.
 - a. En la primera el número recibido como parámetro.
 - b. En la segunda cada número del 1 al 9 según la fila.
 - c. En la tercera el resultado de multiplicar las columnas anteriores

```
createtable TEMP (
    numero int,
    multiplicador int,
    resultado int);
```

2. Inserte la tabla de multiplicar del número proporcionado en la tabla creada y después la muestre por pantalla.

`call tablaMultiplicar(3);`

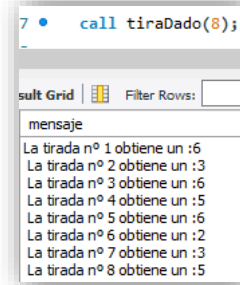
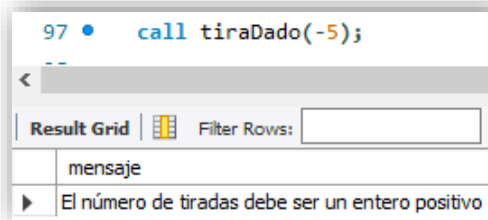
numero	multiplicador	resultado
3	1	3
3	2	6
3	3	9
3	4	12
3	5	15
3	6	18
3	7	21
3	8	24
3	9	27
3	10	30

DIFICULTAD BAJA

2.3.1.5 Crea un procedimiento llamado “tiraDado” que reciba como parámetro un número entero y genere ese número de tiradas de un dado. Para simular la tirada de un dado usaremos unas funciones que en cada ejecución nos permiten obtener un valor aleatorio entre 1 y 6.

- ✓ **FLOOR (RAND () * 6 + 1)** devuelve, para cada ejecución, un número entero aleatorio entre 1 y 6.

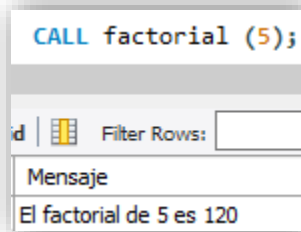
También hará la comprobación de que se recibe un número positivo.



DIFICULTAD BAJA

2.3.1.6 Crear un procedimiento llamado “factorial” que reciba como parámetro un número entero (numero) y te calcule el desarrollo del factorial (numero!).

- ✓ El factorial de un número n ($n!$) es $n! = 1 * 2 * 3 * \dots * n-1 * n$.
- ✓ Ejemplo: El factorial de 5 es $5! = 5 * 4 * 3 * 2 * 1 = 120$.

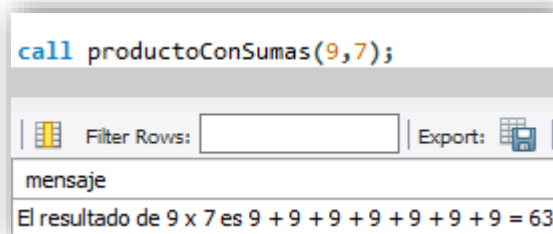


DIFICULTAD BAJA

2.3.1.7 Crea un procedimiento llamado “productoConSumas” que calcule el producto de dos números enteros usando sumas. Los valores de los dos enteros se proporcionarán como parámetros.

- ✓ Ejemplo: Tienes que calcular 3×4 como $3+3+3+3$.
- ✓ No puedes usar el operador de multiplicación (*).
- ✓ Muestra el resultado por pantalla de la siguiente manera:

Mensaje: El resultado de 3×4 es $3+3+3+3=12$.

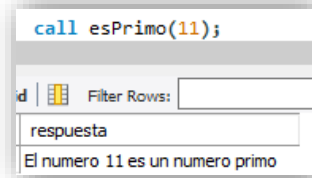


DIFICULTAD MEDIA

2.3.1.8 Crea un procedimiento llamado “esPrimo” que evalúe si un número positivo es o no primo. El valor del número a evaluar se proporcionará como parámetro.

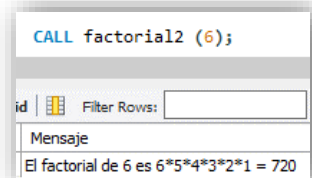
Recordad que un número primo es aquel que solo es divisible entre 1 y sí mismo.

- ✓ El operador % calcula el resto de una división entera.
- ✓ Muestra el resultado por pantalla.



DIFICULTAD MEDIA

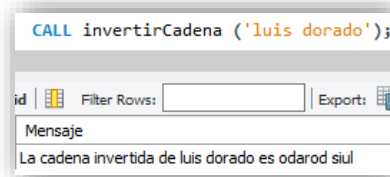
2.3.1.9 Crear un procedimiento llamado “factorial2” que reciba como parámetro un número entero (numero) y te calcule el desarrollo del factorial (numero!) mostrando el resultado con el siguiente formato:



DIFICULTAD MEDIA

2.3.1.10 Crear un procedimiento llamado “invertirCadena” que, dado un parámetro de cadena de caracteres, almacene en una variable esa cadena “al revés”, es decir, con el orden de sus caracteres invertido. Después, muestra el valor de la variable (la cadena invertida).

- ✓ **Ojo:** Solo puedes usar las siguientes funciones: **CONCAT()**, **SUBSTRING(cadena, posición, numCaracteres)** y **CHAR_LENGTH()**.



DIFICULTAD MEDIA

2.3.1.11 Crear un procedimiento llamado “creaVacaciones” que reciba como parámetro un año (tipo YEAR) y que inserte en una tabla todas las fechas de los fines de semana y del mes de agosto.

1. El procedimiento borra la tabla si es que existe y crea la siguiente tabla:

```
CREATE TABLE Vacaciones
(Dia_Vac DATE UNIQUE NOT NULL) ;
```

2. Vamos recorriendo todas las fechas del año proporcionado para incluir todas las que pertenezcan a fin de semana o a agosto.
3. Para terminar: El procedimiento mostrará todas las fechas de la tabla incluyendo una columna con el nombre del día de la semana.
 - ✓ **Pista 1:** Las fechas también se pueden construir como cadenas y se les puede aplicar CONCAT().
 - ✓ **Pista 2:** La función `DATE_ADD (fecha, INTERVAL numDias DAY)` incrementa una fecha un número determinado de días y luego

devuelve la fecha. Ejemplo:

```
DATE_ADD("2017-06-15", INTERVAL 10 DAY)
2017-06-25
```

call creaVacaciones(1990):

Fecha	Día semana
1990-07-14	Saturday
1990-07-15	Sunday
1990-07-21	Saturday
1990-07-22	Sunday
1990-07-28	Saturday
1990-07-29	Sunday
1990-08-01	Wednesday
1990-08-02	Thursday

DIFICULTAD MEDIA

2.3.1.12 Crear un procedimiento llamado “cuentaPalabras” que, dado un parámetro de cadena de caracteres, cuente el número de palabras que contiene.

- ✓ Una palabra está delimitada por un espacio, un inicio de cadena o un fin de cadena.
- ✓ **Ojo:** Solo puedes usar las siguientes funciones: `CONCAT()`, `SUBSTRING(cadena, posición, numCaracteres)` y `CHAR_LENGTH()`.
- ✓ Pista: Para obtener el carácter número 5 de la cadena str tienes que ejecutar `SUBSTRING (str , 5 , 1)`. Equivale a “dame un carácter en la posición 5 de la cadena str.

CALL cuentaPalabras (' luis dorado')

Resultado:
La cadena "luis dorado" tiene 2 palabras

CALL cuentaPalabras ('')

Resultado:
La cadena "" tiene 0 palabras

DIFICULTAD ALTA

2.4 Estructuras de control iterativas anidadas

2.4.1 Sin base de datos

2.4.1.1 Crear un procedimiento llamado “numerosFactoriales” que, dado un parámetro ‘limite’, muestre el resultado de todos factoriales hasta el límite dado por el parámetro.

- ✓ Ejecución: CALL numerosFactoriales (7);
- ✓ Respuesta por pantalla:

1! = 1
 2! = 2
 3! = 6
 4! = 24
 5! = 120
 6! = 720
 7! = 5040

DIFICULTAD BAJA

2.4.1.2 Crear un procedimiento llamado “numerosPrimos” que, dado un parámetro ‘cantidad’, muestre n-primeros números primos.

- ✓ Ejecución: CALL numerosPrimos (20); Mostrará los 20 primeros números primos.
- ✓ Respuesta por pantalla:

Los 20 primeros números primos son: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67 y 71.

DIFICULTAD MEDIA

2.4.1.3 Crear un procedimiento “creaRampa” Dado como parámetro un número que debe ser menor que 30 construye la siguiente estructura del ejemplo hasta el límite que marque el parámetro.

- ✓ Por ejemplo, para el parámetro 7 se mostrará por pantalla.

```
1
22
333
4444
55555
666666
7777777
```

- ✓ Recuerda que para hacer un salto de línea se debe usar el “\n”.

call creaRampa(10);

id	mensaje
1	1
2	2 2
3	3 3 3
4	4 4 4 4
5	5 5 5 5 5
6	6 6 6 6 6 6
7	7 7 7 7 7 7 7
8	8 8 8 8 8 8 8 8
9	9 9 9 9 9 9 9 9 9
10	10 10 10 10 10 10 10 10 10 10

DIFICULTAD MEDIA

2.4.1.4 Crear un procedimiento “creaFlecha” Dado como parámetro un número que debe ser menor que 30 construye la siguiente estructura del ejemplo hasta el límite que marque el parámetro.

- ✓ Por ejemplo, para el parámetro 7 se mostrará por pantalla.

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
7 7 7 7 7 7 7
6 6 6 6 6 6
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```

- ✓ Recuerda que para hacer un salto de línea se debe usar el “\n”.

call creaFlecha(6);

id	mensaje
1	1
2	2 2
3	3 3 3
4	4 4 4 4
5	5 5 5 5 5
6	6 6 6 6 6 6
5	5 5 5 5 5
4	4 4 4 4
3	3 3 3
2	2 2
1	1

DIFICULTAD ALTA

2.5 Funciones

2.5.1 BD Empresa

2.5.1.1 Crea una función 'fn_diferenciaSalario' que reciba un código de empleado y que calcule la diferencia de salario de un empleado con el salario medio de toda la empresa (redondeado a dos decimales).

- ✓ Una vez creada, úsala para mostrar la lista de nombres y apellidos de empleados junto a su salario y la diferencia con el salario medio.

The screenshot shows a SQL query window with the following text:

```
SELECT nombre, apellidos, salario, fn_diferenciaSalario(cod_empleado)
FROM empleados
ORDER BY fn_diferenciaSalario(cod_empleado) DESC;
```

Below the query, a table of results is displayed with the following data:

nombre	apellidos	salario	fn_diferenciaSalario(cod_empleado)
ANTONIO	LOPEZ	1720	417.06
MARCOS	PEREZ	1480	177.06
ADRIANA	ALBA	1450	147.06
AURELIO	CAMPS	1450	147.06
JULIO	PEREZ	1440	137.06
AUGUSTO	CARCTA	1430	117.06

DIFICULTAD BAJA

2.5.1.2 Crea una función 'fn_tipoDirector' que reciba el código de un empleado y el numero de un departamento, y devuelva:

- ✓ El texto 'No es director' si no es director de ese departamento
- ✓ El texto 'Director' en propiedad' si es director en propiedad (P) de ese departamento
- ✓ El texto 'Director' en funciones' si es director en funciones (F) de ese departamento.

Una vez creada, úsala para mostrar el nombre, apellidos, oficio, departamento que dirigen (si no dirigen ninguno, mostrar valor nulo) y el tipo de director que es (utiliza la función tipoDirector).

The screenshot shows a SQL query window with the following text:

```
SELECT e.nombre, e.apellidos, e.oficio, e.num_departamento, fn_tipoDirector(e.cod_empleado, e.num_departamento)
FROM empleados e;
```

Below the query, a table of results is displayed with the following data:

nombre	apellidos	oficio	num_departamento	fn_tipoDirector(e.cod_empleado, e.num_departamento)
OSCAR	PONS	PRESIDENTE	121	No es director
MARIO	LARGA	JEFESECTOR	112	No es director
LUIGIANO	TEROL	JEFESECTOR	112	No es director
LUIGI	PEREZ	JEFESECTOR	121	Director en funciones

DIFICULTAD BAJA

2.5.1.3 Crea una función 'fn_presupuestoPorEmpleado' que calcule el presupuesto por empleado de un departamento (presupuesto del departamento / número de empleados).

- ✓ Una vez creada, úsala para mostrar la lista de nombres de departamento junto a su presupuesto y su presupuesto por empleado.

The screenshot shows a SQL query: `SELECT nombre, presupuesto, fn_presupuestoPorEmpleado(num_departamento) FROM departamentos;`. The function name is highlighted with a red box. Below the query, a table displays the results:

nombre	presupuesto	fn_presupuestoPorEmpleado(num_departamento)
DIRECCION GENERAL	12000	4000
DIRECC. COMERCIAL	15000	5000
SECTOR INDUSTRIAL	11000	1375

DIFICULTAD BAJA

2.5.2 BD Jardinería

2.5.2.1 Escribe una función llamada “fn_numPedidosPorEstado” que reciba como parámetro un estado de pedido y devuelva cuántos pedidos de ese estado se tiene.

- ✓ Úsala en una consulta select como se muestra en el ejemplo.

The screenshot shows a SQL query: `select estado, fn_numPedidosPorEstado(estado) from pedidos group by estado;`. Below the query, a table displays the results:

estado	fn_numPedidosPorEstado(estado)
Entregado	61
Rechazado	24
Pendiente	29
Pendiente	1

DIFICULTAD BAJA

2.5.2.2 Escribe una función llamada “fn_numClientesGama” que reciba como parámetro la gama de un producto y devuelva el N° clientes que han pedido productos de esa gama.

- ✓ Úsala en una consulta select como se muestra en el ejemplo.

The screenshot shows a SQL query: `select gama, fn_numClientesGama(gama) from productos group by gama;`. Below the query, a table displays the results:

gama	fn_numClientesGama(gama)
Aromáticas	29
Frutales	122
Herramientas	28
Ornamentales	113

DIFICULTAD BAJA

2.5.2.3 Escribe una función llamada “fn_numClientesOficinaPais” que reciba como parámetro un país y devuelva el N° de clientes cuya oficina (la de su representante) esté localizada en ese país.

- ✓ Úsala en una consulta SELECT como se muestra en el ejemplo.

```
select pais, fn_numClientesSinOficina(pais) from oficinas group by pais;
```

id	Grid	Filter Rows:	Export:	Wrap Cell Content:
pais	fn_numClientesSinOficina(pais)			
España	21			
EEUU	6			
Inglaterra	0			
Francia	2			
Australia	7			
Japón	0			

DIFICULTAD BAJA

- 2.5.2.4** Escribe una función llamada “fn_OficinaConMasEmpleados” que no reciba parámetros y devuelva el código de oficina que más empleados tiene. Si fueran varias, devolver la primera de la lista.

```
select CONCAT('La oficina con más empleados es: ',fn_OficinaConMasEmpleados()) as Mensaje;
```

Grid	Filter Rows:	Export:	Wrap Cell Content:
Mensaje			
La oficina con más empleados es: TAL-ES			

DIFICULTAD MEDIA

- 2.5.2.5** Escribe una función llamada “fn_ProductoMaxPedido” que no reciba parámetros y devuelva el nombre del producto del que más unidades se hayan vendido en un mismo pedido. Si fueran varios, devolver la primera de la lista.

```
select fn_ProductoMaxPedido();
```

id	Filter Rows:
fn_ProductoMaxPedido()	
Thymus Citriodora (Tomillo limón)	

DIFICULTAD MEDIA

2.5.3 Sin base de datos

- 2.5.3.1** Escribe una función llamada *fn_hipotenusa* que devuelva el valor de la hipotenusa de un triángulo a partir de los valores de sus catetos recibidos como parámetros.

- ✓ **POWER(X,Y)** retorna el valor de X a la potencia de Y.
- ✓ **SQRT(X)** Retorna la raíz cuadrada de un número no negativo X.

`call sp_hipotenusa(3,2);`

rid	Filter Rows:
Mensaje	
	3.60555

DIFICULTAD BAJA

2.5.3.2 Escribe una función llamada

fn_areaCirculo que devuelva el valor del área de un círculo a partir del valor del radio que recibirá como parámetro de entrada.

- ✓ Para comprobar que la función es correcta, ejecuta un procedimiento que muestre por pantalla el valor de retorno de la función.

`call sp_areaCirculo(3);`

rid	Filter Rows:
Mensaje	
	28.2744

DIFICULTAD BAJA

2.5.3.3 Escribe una función llamada

fn_diaSemana que reciba como parámetro de entrada un valor numérico que represente un día de

la semana y que devuelva una cadena de caracteres con el nombre del día de la semana correspondiente.

Por ejemplo, para el valor de entrada 1 debería devolver la cadena **lunes**.

- ✓ Para comprobar que la función es correcta, ejecuta un procedimiento que muestre por pantalla el valor de retorno de la función.

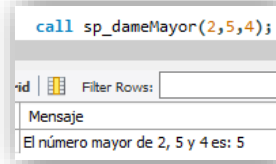
`call sp_diaSemana(5);`

rid	Filter Rows:
Mensaje	
	Viernes

DIFICULTAD BAJA

2.5.3.4 Escribe una función llamada *fn_dameMayor* que reciba tres números reales como parámetros de entrada y devuelva el mayor de los tres.

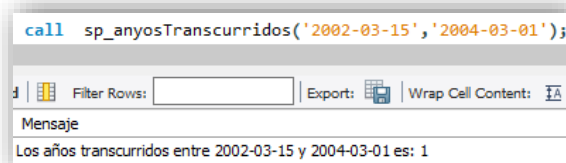
- ✓ Para comprobar que la función es correcta, ejecuta un procedimiento que muestre por pantalla el valor de retorno de la función.



DIFICULTAD BAJA

2.5.3.5 Escribe una función llamada *fn_anyosTrancurridos* que devuelva como salida el número de años que han transcurrido entre dos fechas que se reciben como parámetros de entrada. Por ejemplo, si pasamos como parámetros de entrada las fechas 2008-06-17 y 2018-05-20 la función tiene que devolver que han pasado 9 años.

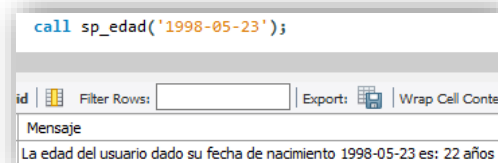
- ✓ Para realizar esta función puede hacer uso de las siguientes funciones que nos proporciona MySQL: *TIMESTAMPDIFF*.
- ✓ Para comprobar que la función es correcta, ejecuta un procedimiento que muestre por pantalla el valor de retorno de la función.



DIFICULTAD BAJA

2.5.3.6 Escribe una función llamada *fn_edad* que dada una fecha de nacimiento calcule la edad.

- ✓ Para comprobar que la función es correcta, ejecuta un procedimiento que muestre por pantalla el valor de retorno de la función.

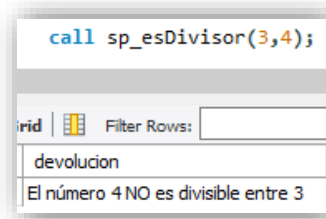


DIFICULTAD BAJA

2.5.3.7 Implementar una función *fn_esDivisor* que reciba como parámetro dos números enteros (*a* y *b*) que compruebe si *a* es divisor que *b* y devuelva un entero que indique el resultado (1 si el número *a* es divisor de *b* y 0 en caso contrario).

- ✓ La función no debe mostrar nada, solo devolver un entero.
- ✓ Si *a* es mayor que *b* entonces devolverá -1 (error).

- ✓ Para comprobar que la función es correcta, ejecuta un procedimiento que muestre por pantalla el valor de retorno de la función.



DIFICULTAD BAJA

- 2.5.3.8** Crear un procedimiento llamado “factorialesConFuncion” que, dado un parámetro ‘limite’, muestre el resultado de todos factoriales hasta el límite dado por el parámetro. **Es este caso usarás una función “fn_factorial” que dado un número te devuelva su factorial.**

- ✓ Ejecución: CALL factorialesConFuncion (7);
- ✓ Respuesta por pantalla:

1! = 1
 2! = 2
 3! = 6
 4! = 24
 5! = 120
 6! = 720
 7! = 5040

DIFICULTAD BAJA

- 2.5.3.9** O esCrear un procedimiento llamado “primosConfuncion” que, dado un parámetro ‘cantidad’, muestre n-primeros números primos. **Es este caso usarás una función “fn_esPrimo” que dado un número te devuelva un valor booleano que indique si es primo o no.**

- ✓ Ejecución: CALL numerosPrimos (20); Mostrará los 20 primeros números primos.
- ✓ Respuesta por pantalla:

Los 20 primeros números primos son: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67 y 71.

DIFICULTAD BAJA

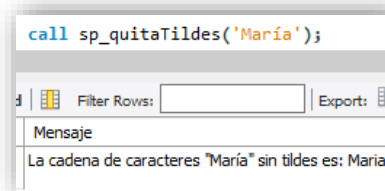
2.5.3.10 Escribe una función llamada

fn_quitaTildes que reciba una cadena de entrada y devuelva la misma cadena, pero sin acentos.

La función tendrá que reemplazar todas las vocales que tengan un acento por la misma vocal, pero sin tilde.

Por ejemplo, si la función recibe como parámetro de entrada la cadena *María*, la función debe devolver la cadena *Maria*.

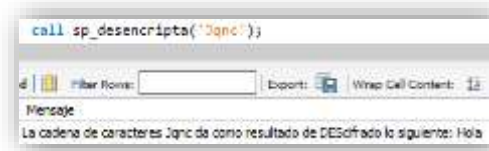
- ✓ Pista: Para obtener el carácter número 5 de la cadena *str* tienes que ejecutar **SUBSTRING (str , 5 , 1)**. Equivale a “dame **un** carácter en la posición **5** de la cadena **str**”.
- ✓ Para comprobar que la función es correcta, ejecuta un procedimiento que muestre por pantalla el valor de retorno de la función.

**DIFICULTAD MEDIA**

2.5.3.11 Escribe una función llamada *fn_encrypta* que se encargue de encriptar un mensaje que recibe como parámetro, para ello se sumará 2 al código ASCII del carácter (función de sistema **ASCII (carácter)**) para después transformar ese código ASCII incrementado a carácter otra vez (función de sistema **CHAR (codigoASCII)**).

Escribe también la función “*fn_descripta*” que descripte el mensaje anterior.

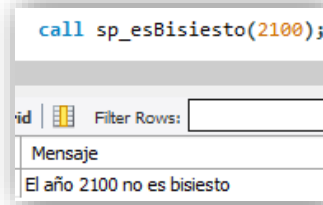
- ✓ Pista: Para obtener el carácter número 5 de la cadena *str* tienes que ejecutar **SUBSTRING (str , 5 , 1)**. Equivale a “dame **un** carácter en la posición **5** de la cadena **str**”.
- ✓ Para comprobar que las funciones son correctas, ejecuta un procedimiento que muestre por pantalla el valor de retorno de la función.

**DIFICULTAD MEDIA**

2.5.3.12 Función llamada “*fn_esBisiesto*” que a partir de un año retorne true si es bisiesto, false si no lo es.

- ✓ Nota: Los años bisiestos son los múltiplos de 4 que no son múltiplos de 100, pero los múltiplos de 400 sí son bisiestos.

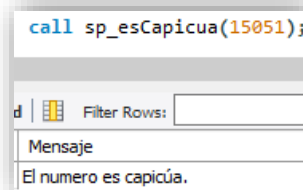
- ✓ Ejemplo: 1900 no fue bisiesto, 2000 si lo fue y 2100 no lo será.
- ✓ Para comprobar que la función es correcta, ejecuta un procedimiento que muestre por pantalla el valor de retorno de la función.



DIFICULTAD MEDIA

2.5.3.13 Escribe una función llamada “fn_esCapicua” que recibiendo un número retorne true si el N° es capicúa y false si no lo es.

- ✓ Primero debes construir, con operaciones matemáticas, un número con las posiciones de sus cifras invertidas. Después compararlo con el número pasado por parámetro.
- ✓ Para comprobar que la función es correcta, ejecuta un procedimiento que muestre por pantalla el valor de retorno de la función.



DIFICULTAD ALTA

2.6 Cursores

2.6.1 BD Empresa

2.6.1.1 Crea un procedimiento denominado 'revisarSalarios' que actualice el salario de todos los empleados según su oficio de acuerdo a las siguientes reglas:

- ✓ JEFESECTOR: incrementar 5% el salario
- ✓ VENDEDOR: incrementar 2% el salario
- ✓ ANALISTA: incrementar 7% el salario
- ✓ EMPLEADO: incrementar 1% el salario

Usa un cursor que recorra TODOS los empleados y que vaya actualizando, uno a uno, cada empleado según su oficio.

DIFICULTAD BAJA

2.6.1.2 Crea un procedimiento denominado 'revisarPresupuestosCentro' que actualice el presupuesto de todos los departamentos de un centro (cuyo identificador se pasará por parámetro) según el tipo de trabajadores que tiene siguiendo las siguientes reglas:

- ✓ 5000€ por cada JEFESECTOR
- ✓ 2500€ por cada ANALISTA
- ✓ 3000€ por cada VENDEDOR
- ✓ 1500€ por cada EMPLEADO

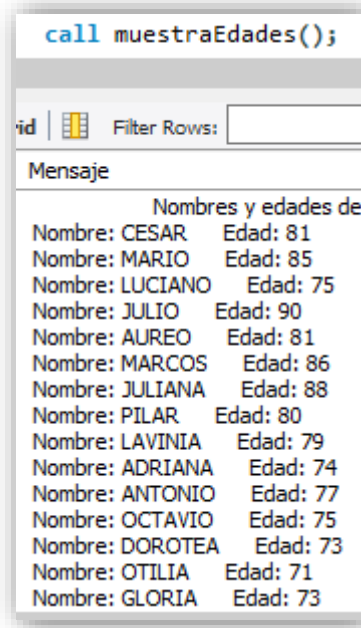
Ejemplo: A un departamento con 2 empleados y 3 vendedores se le asignará $2 \times 1500 + 3 \times 3000 = 12000\text{€}$.

Notas: Para recorrer los departamentos de un centro usaremos un cursor y para hallar el número de trabajadores de cada tipo usaremos COUNT(*).

DIFICULTAD BAJA

2.6.1.3 Escribir un procedimiento denominado 'muestraEdades' que recorra la tabla empleados mostrando por cada empleado su edad.

Nota: Hacer uso para ello de la función 'fn_edad' anterior que calcula la edad a partir de una fecha de nacimiento.



Mensaje	
Nombres y edades de	
Nombre: CESAR	Edad: 81
Nombre: MARIO	Edad: 85
Nombre: LUCIANO	Edad: 75
Nombre: JULIO	Edad: 90
Nombre: AUREO	Edad: 81
Nombre: MARCOS	Edad: 86
Nombre: JULIANA	Edad: 88
Nombre: PILAR	Edad: 80
Nombre: LAVINIA	Edad: 79
Nombre: ADRIANA	Edad: 74
Nombre: ANTONIO	Edad: 77
Nombre: OCTAVIO	Edad: 75
Nombre: DOROTEA	Edad: 73
Nombre: OTILIA	Edad: 71
Nombre: GLORIA	Edad: 73

DIFICULTAD BAJA

2.6.2 BD Pedidos

2.6.2.1 Escribe un procedimiento que revise las ventas de las oficinas. Para cada oficina calcula la suma de las ventas de sus empleados y actualiza el valor de ventas de la oficina con el nuevo valor.

DIFICULTAD BAJA

- 2.6.2.2** Escribe un procedimiento que reciba una fábrica como parámetro de entrada. Revisa todos sus productos. Para cada uno de ellos, si el producto se ha vendido más de 3 veces, aumentarle el precio un 5%. Si no, rebajarle el precio un 10%.

DIFICULTAD BAJA

- 2.6.2.3** Escribe un procedimiento que revise las ventas realizadas por cada empleado. Si las ventas no superan el objetivo, reduciremos su objetivo en 1000€.

DIFICULTAD BAJA

- 2.6.2.4** Crea una nueva tabla denominada 'clientesMorosos' con los campos numCliente y nombre. Escribe un procedimiento que revise los pedidos realizados por cada cliente. Si el importe total de los pedidos de un cliente supera su límite de crédito, añádelo a la tabla de cliente morosos.

DIFICULTAD BAJA

2.6.3 BD Tiendas

- 2.6.3.1** Crear un procedimiento llamado 'incrementaPrecio' que aumente precio de venta en un 5% de los artículos cuyo id de fabricante se pasa por parámetro. Solo se pueden hacer UPDATEs de una sola fila cada vez.

DIFICULTAD BAJA

- 2.6.3.2** Crear un procedimiento llamado 'suministrarTiendas' que, dado un fabricante, abastezca con 5 unidades de todos los productos de ese fabricante a todas las tiendas. Es decir, debes insertar pedidos de 5 unidades de cada producto del fabricante a todas las tiendas.

DIFICULTAD MEDIA

- 2.6.3.3** Crear un procedimiento llamado 'insertaPedido' que recibirá dos parámetros: un identificador de fabricante y un código postal.

Para cada una de las tiendas que tengan el código postal especificado, inserta un pedido de una unidad de cada uno de los artículos del fabricante especificado.

DIFICULTAD MEDIA

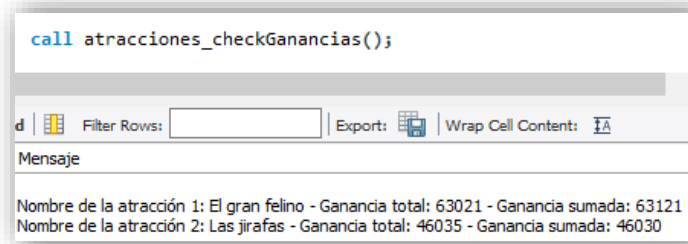
2.6.4 BD Circo

- 2.6.4.1** Crea un procedimiento de nombre “*atracciones_checkGanancias*” en el que queramos comprobar si el valor del campo “ganancias” de cada atracción coinciden con la suma de las ganancias de los días en los que se celebró. Para las atracciones que no cumplen que la suma de las ganancias de cada día sea igual a las ganancias totales. El procedimiento debe devolver una cadena con el formato:

Nombre de la atracción 1: nombre - Ganancia total: XX - Ganancia sumada: XX

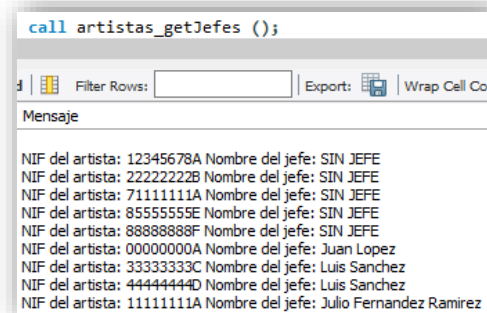
Nombre de la atracción 2: nombre - Ganancia total: XX - Ganancia sumada: XX

...



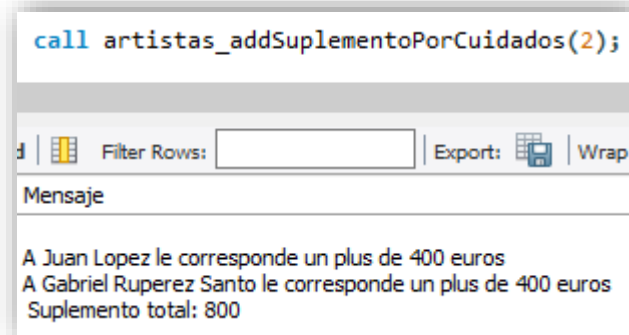
DIFICULTAD BAJA

- 2.6.4.2** Crea un procedimiento sin parámetros de nombre “*artistas_getJefes*” que muestre a cada artista (NIF) con su jefe (nombre, apellidos). En caso de que no tenga jefe deberá mostrar la cadena SIN JEFE. Vas a crear un cursor que recorra todos los artistas (que consulta asociada solo use la tabla “artistas”) y por cada artista que busque su jefe.



DIFICULTAD BAJA

- 2.6.4.3** Crea un procedimiento de nombre “*artistas_addSuplementoPorCuidados*”, que compruebe a cuantos animales cuida cada uno de los artistas. Aquellos artistas que cuidan más de un número de animales indicados por el parámetro se les aplicará un plus a su nómina igual al número de animales que cuida multiplicado por 100 euros. Muestra el nombre y complemento que se le aplicaría a cada artista, así como la suma de todos los complementos.

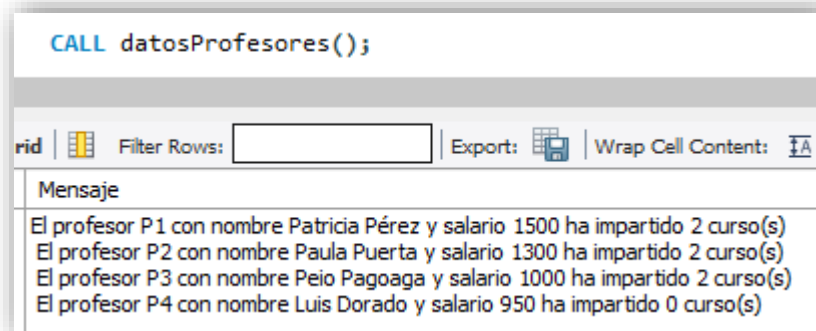


DIFICULTAD MEDIA

2.6.5 BD Formación

- 2.6.5.1** Crear un procedimiento llamado 'datosProfesores' que muestre para cada profesor su código, nombre, salario y la cantidad de cursos impartidos de la manera que se muestra en el ejemplo.

✓ OJO: si un profesor no imparte ningún curso, debe aparecer con 0 cursos.



DIFICULTAD BAJA

- 2.6.5.2** Ampliar el ejercicio anterior y nombrarlo como 'datosProfesores2' para que para que debajo de cada línea de profesor, en un segundo nivel tabulado (con espacios), aparezca el código, nombre del curso y las horas del curso o cursos que imparte.

✓ Ejemplo:

CALL datosProfesores2();

Filter Rows: Export: Wrap Cell Content:

Mensaje
El profesor P1 con nombre Patricia Pérez y salario 1500 ha impartido 2 curso(s) Curso: SO Horas: 300 Curso: HTML Horas: 200
El profesor P2 con nombre Paula Puerta y salario 1300 ha impartido 2 curso(s) Curso: CONTABILIDAD Horas: 200 Curso: FOL Horas: 200
El profesor P3 con nombre Peio Pagoaga y salario 1000 ha impartido 2 curso(s) Curso: HW Horas: 150 Curso: ASBD Horas: 150
El profesor P4 con nombre Luis Dorado y salario 950 ha impartido 0 curso(s)



Pista: Algoritmo

1. Creamos un cursor y le asociamos la consulta de todos los profesores y el número de cursos que han impartido.
2. Creamos otro cursor y le asociamos la consulta de todos los cursillos de un profesor (variable).
3. Abrimos el cursor de profesores y cargamos la primera fila.
4. Mientras la última carga del profesor se halla realizado con éxito:
 - a.
 - b. Mostramos info del profesor.
 - c. Abrimos el cursor para los cursos del profesor de la fila que hemos cargado actualmente. Cargamos la primera fila del curso.
 - d. Mientras la última carga del profesor se halla realizado con éxito:
 - i. Mostramos info de ese curso
 - ii. Intentamos obtener una nueva fila de un curso de ese profesor
 - iii. Volvemos a **d**
 - e. Cerramos el cursor de cursos de ese profesor.
 - f. Intentamos obtener una nueva fila de un profesor.
 - g. Volvemos a **2**
5. Cerramos el cursor de profesores

DIFICULTAD MEDIA

- 2.6.5.3** Ampliar el ejercicio anterior y nombrarlo como 'datosProfesores3' para que para que debajo de cada línea de cursillo, en un tercer nivel tabulado (con espacios), aparezca el nombre de los alumnos que lo cursan así como la calificación que han obtenido.

✓ Ejemplo:

CALL datosProfesores3();	
id	Filter Rows: <input type="text"/> Export:  Wrap Cell Content: 
Mensaje	
El profesor P1 con nombre Patricia Pérez y salario 1500 ha impartido 2 curso(s)	
Curso: SO Horas: 300	
Alumno: Antonio Antúnez Nota: 9.0	
Alumno: Amalia Naya Nota: 9.0	
Curso: HTML Horas: 200	
El profesor P2 con nombre Paula Puerta y salario 1300 ha impartido 2 curso(s)	
Curso: CONTABILIDAD Horas: 200	
Curso: FOL Horas: 200	
El profesor P3 con nombre Peio Pagoaga y salario 1000 ha impartido 2 curso(s)	
Curso: HW Horas: 150	
Alumno: Ane Aranburu Nota: 8.0	
Alumno: Amalia Naya Nota: 9.0	
Curso: ASBD Horas: 150	
El profesor P4 con nombre Luis Dorado y salario 950 ha impartido 0 curso(s)	

DIFICULTAD ALTA

3 PROGRAMACIÓN DE BBDD EN MYSQL III

3.1 Control de errores

3.1.1 Sin base de datos (Captura de excepciones de sistema)

3.1.1.1 Crea una **base de datos** llamada `test` que contenga una **tabla** llamada `alumno`. La tabla debe tener cuatro columnas:

- ✓ `id`: enteros insigno (clave primaria).
- ✓ `nombre`: cadena de 50 caracteres.
- ✓ `apellido1`: cadena de 50 caracteres.
- ✓ `apellido2`: cadena de 50 caracteres.

Una vez creada la base de datos y la tabla deberá **crear una función** llamada `fn_insertar_alumno` con las siguientes características:

- El procedimiento recibe cuatro parámetros (`id`, `nombre`, `apellido1`, `apellido2`) y los insertará en la tabla `alumno`.
- La función devolverá 0 si la operación se ha podido realizar con éxito y un valor igual a -1 si hay una infracción de clave primaria y un valor igual a -2 si la tabla no existe.
- Habrá dos manejadores, uno para cada error: 1062 infracción de clave primaria y 1146 tabla no existente.

```
-- Primera vez que ejecutamos esta llamada existiendo la tabla
select fn_insertar_alumno(3,'Vanesa', 'Dorado','Lopez');
```

id
fn_insertar_alumno(3,'Vanesa', 'Dorado','Lopez')
0

```
-- Segunda vez que ejecutamos esta llamada existiendo la tabla
select fn_insertar_alumno(3,'Vanesa', 'Dorado','Lopez');
```

Grid
fn_insertar_alumno(3,'Vanesa', 'Dorado','Lopez')
-1

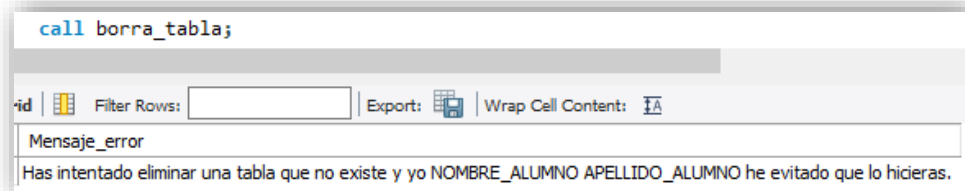


DIFICULTAD BAJA

3.1.2 BD Universidad (Captura de excepciones de sistema)

3.1.2.1 Realizar un procedimiento llamado “borra_tabla” que intente eliminar la tabla pruebaUniversidad (esta tabla no existe).

- ✓ Esto debe provocar un error 1051 o SQLSTATE '42S02' que debemos tratar saliendo del procedimiento y mostrando el siguiente mensaje:
"Has intentado eliminar una tabla que no existe y yo NOMBRE_ALUMNO APELLIDO_ALUMNO he evitado que lo hicieras."
➤ Sustituye NOMBRE_ALUMNO APELLIDO_ALUMNO por tu nombre.
- ✓ Si la tabla existiera debería eliminarla y, después, mostrar un mensaje en el que se indique lo siguiente: "Tabla eliminada con éxito"

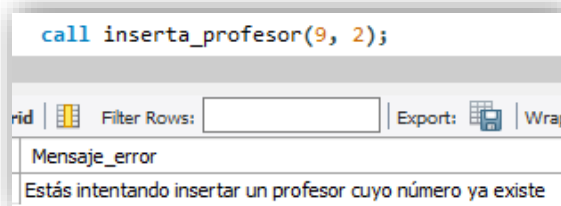
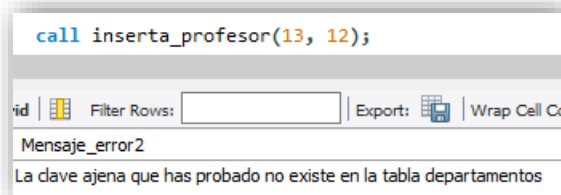
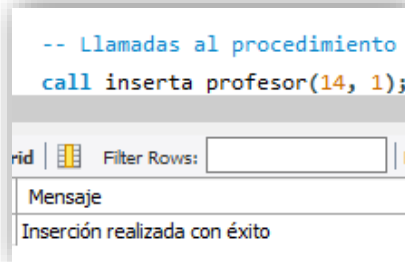


DIFICULTAD BAJA

3.1.2.2 Realizar un procedimiento llamado “inserta_profesor” que debe permitir añadir un nuevo profesor en la tabla profesor, para ello hay que incluir su identificador de profesor y su identificador de departamento.

- ✓ Si el profesor que se intentara insertar en la tabla ya estuviera en dicha tabla debe generar un error 1062 o SQLSTATE 'S1009' o '23000' que debemos tratar saliendo del procedimiento y mostrando el siguiente mensaje: "Has intentado insertar un profesor cuyo número ya existe"
- ✓ Si el número de departamento pasado como parámetro no existiera debería generar un error 1452 o SQLSTATE '23000' que debemos tratar saliendo del procedimiento y mostrando el siguiente mensaje: "La clave ajena que has probado no existe en la tabla departamentos"

- ✓ Si la inserción es correcta debe realizarla y, después, mostrar el siguiente mensaje: "Inserción realizada con éxito".



DIFICULTAD BAJA

3.1.2.3 Realizar un procedimiento llamado "actualiza_asignatura" que actualice todos los valores de una asignatura. Para ello se pasarán todos los valores de los campos de la tabla (identificador, nombre, créditos, tipo, curso, cuatrimestre, idProfesor e idGrado).

- ✓ Si el tipo de la asignatura que se intenta actualizar no corresponde con los que están en el campo enumerado debe generar un error 1265 o SQLSTATE '01000' que debemos tratar saliendo del programa y mostrando el siguiente mensaje: 'Has intentado actualizar con un tipo de asignatura que no existe, solo se permite "básica", "obligatoria" u "optativa"'
- ✓ Si el identificador de profesor o el de grado pasado como parámetro no existiera debería generar un error 1452 o SQLSTATE '23000' que debemos tratar saliendo del procedimiento y mostrando el siguiente mensaje: 'La clave ajena que has probado no existe en la tabla correspondiente'.

- ✓ Finalmente, si la actualización se realiza con éxito debe mostrar actualizar y mostrar el siguiente mensaje: "La actualización se ha realizado correctamente".

```
call actualiza_asignatura(2, 'Cálculo', 6, 'básica', 1, 1, 9, 4);
```

id	Filter Rows:	Export:	Wrap Cell Content:
mensaje			
La actualización se ha realizado correctamente			

```
call actualiza_asignatura(2, 'Cálculo', 6, 'básica', 1, 1, 93, 4);
```

id	Filter Rows:	Export:	Wrap Cell Content:
Mensaje_error			
La clave ajena que has probado no existe en la tabla grado o profesor			

```
call actualiza_asignatura(2, 'Cálculo', 6, 'elemental', 1, 1, 9, 4);
```

	Filter Rows:	Export:	Wrap Cell Content:
Mensaje_error			
Has intentado actualizar con un tipo de asignatura que no existe, solo se permite "básica", "obligatoria" u "optativa"			

DIFICULTAD BAJA

3.1.3 BD Circo (Captura de excepciones de sistema y de usuario)

3.1.3.1 Crea un procedimiento de nombre "*animales_add*" que recibe por parámetro los datos de un animal para añadirlo a la tabla ANIMALES. Habrá que tener en cuenta las siguientes condiciones:

- ✓ En caso de intentar dar de alta un animal con el mismo nombre (valor de clave primaria duplicada), captura la excepción y haz que muestre por pantalla el mensaje correspondiente. En este caso no realizará el INSERT.
- ✓ En el caso de que el nombre de la pista o de la atracción no exista, captura la excepción y haz que muestre por pantalla el mensaje correspondiente. En este caso no realizará el INSERT.
- ✓ En caso de que el alta sea correcta realizará el INSERT y mostrará el mensaje de éxito.
- Para saber el número de excepción que tienes que capturar, provoca el fallo y anota el número.
- Si provocamos los errores podemos comprobar que:
 - Error 1062: Clave primaria duplicada
 - Error 1452: Error de clave foránea.

- Ampliación: Crea las siguientes excepciones: `ex_claveDuplicada`, `ex_pista_atracc_no_existe` y úsalas en vez de los errores 1062 y 1452.

```
call animales_add('Leo', 'León', 3, 230, 1.34, null, null); -- Clave duplicada
```

id | Filter Rows: | Export: | Wrap Cell Content: |

Mensaje

El valor de clave primaria Leo ya existe en la tabla

```
call animales_add('Leo2', 'León', 3, 230, 1.34, 'NO EXISTE', null); -- Atracción que no existe
```

id | Filter Rows: | Export: | Wrap Cell Content: |

Mensaje

Violación de clave ajena: la pista o la atracción proporcionadas no existen

```
call animales_add('Leo2', 'León', 3, 230, 1.34, 'El gran felino', 'SUPER'); -- Todo correcto
```

id | Filter Rows: | Export: | Wrap Cell Content: |

Mensaje

La inserción de Leo2 se ha realizado con éxito

DIFICULTAD BAJA

- 3.1.3.2** Crea una función de nombre `fn_artistas_add` que inserta un nuevo artista con todos sus datos como parámetros. Si todo va bien la función insertará el artista y se devolverá un 0.

Se controlarán los siguientes errores:

- Si el artista existe y la inserción nos devuelve un error por infracción de clave primaria (error 1062) la capturaremos y la función devolverá -1.
 - Que no exista ningún artista con el NIF correspondiente al campo `nif_jefe`. En este caso no esperaremos al error de MySQL, sino que comprobaremos si existe el jefe a insertar y, en caso de no existir, lanzaremos una excepción de usuario que capturaremos y la función devolverá -2.
- ✓ Cualquier excepción que se produzca debe hacer que finalice la función (con RETURN) pero no las siguientes instrucciones.
- ✓ **Ampliación:** Crea las excepciones `ex_jefe_no_existe` (excepción de usuario) y `ex_artista_duplicado` (error 1062).

```
SELECT fn_artistas_add('11111111B', 'Orten Sanlat', 'Cristina', '11111111A') AS 'Devolución de función fn_artistas_add()';
```

id | Filter Rows: | Export: | Wrap Cell Content: |

Devolución de función
fn_artistas_add()

0


```
SELECT fn_artistas_add('11111111','Ortem-Seniat','Cristine',null) AS 'Devolución de función fn_artistas_add()'
```

fn_artistas_add()
-1

```
SELECT fn_artistas_add('11111122','Ortem-Seniat','Cristine','11111122') AS 'Devolución de función fn_artistas_add()'
```

fn_artistas_add()
-2

DIFICULTAD BAJA

- 3.1.3.3** Crea un procedimiento de nombre *“animales_Delete”* que, dado el nombre de un animal, lo borre. Antes tendrá que borrar todas las filas relacionadas. En el caso de que el animal tenga menos de 2 años no estará permitido borrarlo y lanzará una excepción de usuario (recordar que el código SQLSTATE para excepciones definidas por el usuario es el 45000) con el texto: *“No es posible dar de baja a animales con menos de dos años”* y un ErrorCode 1644. En el caso de que el animal no exista, deberá lanzar una excepción con el texto 'Ese animal no existe' y un ErrorCode 1643.

Nota 1: En este ejercicio **NO** se pide capturar la excepción con DECLARE ... HANDLER, tan solo lanzarlas con SIGNAL.

Nota 2: Fijarse que este procedimiento borra los datos de varias tablas, por lo que necesitaríamos hacer uso de una transacción como veremos en el siguiente apartado.

```
call animales_delete('no_existe'); -- Devuelve el código 1643
```

Time	Action	Message
1 19:51:15	call animales_delete(no_existe)	Error Code: 1643. Ese animal no existe

```
call animales_delete('Bemí'); -- No cumple que la edad sea superior a 2 años.
```

Time	Action	Message
1 19:51:53	call animales_delete(Bemí)	Error Code: 1644. No se puede dar de baja a animales con menos de dos años



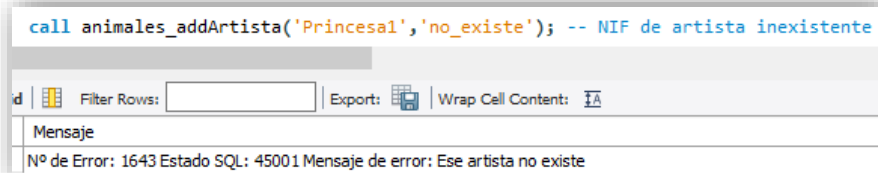
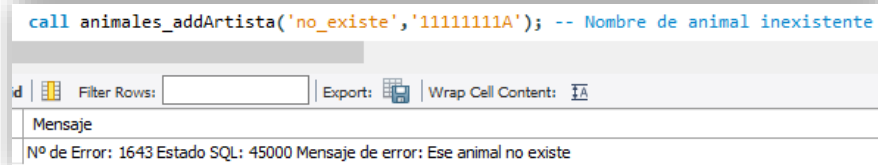
DIFICULTAD BAJA

3.1.3.4 Crea un procedimiento de nombre *“animales_addArtista”* al que se le pase el nombre de un animal y el NIF de un artista y asigne el cuidador al animal. Deberá comprobar:

1. Que el animal y el artista existen. En caso de que no, deberá lanzar una excepción con el ErrorCode 1643 en ambos casos y texto 'El animal no existe' o 'El artista no existe' según el caso.
2. Que el animal no se halla ya asignado al artista. En caso de que no, deberá lanzar una excepción con el ErrorCode Error 1062 y texto 'El animal ya está asignado al artista'.

Nota 1: En este ejercicio **SI** se pide capturar la excepción con un **solo** DECLARE ... HANDLER y lanzar el error con SIGNAL.

Nota 1: Será el HANDLER único el que muestre el mensaje por pantalla obteniendo los datos del error mediante GET DIAGNOSTICS.



DIFICULTAD MEDIA

3.2 Transacciones I

3.2.1 Sin base de datos

3.2.1.1 Ejecuta las siguientes instrucciones y resuelve las cuestiones que se plantean en cada paso.

```

DROPDATABASEIFEXISTS test;
CREATEDATABASE test CHARACTERSET utf8mb4;
USE test;

CREATETABLE producto (
  idINT UNSIGNED AUTO_INCREMENT PRIMARYKEY,
  nombre VARCHAR(100) NOTNULL,
  precio DOUBLE
);

INSERTINTO producto (id, nombre) VALUES (1, 'Primero');
INSERTINTO producto (id, nombre) VALUES (2, 'Segundo');
INSERTINTO producto (id, nombre) VALUES (3, 'Tercero');

```

```

-- 1. ¿Qué devolverá esta consulta? ¿Por qué?
SELECT*
FROM producto;

```

```

-- 2. Vamos a intentar deshacer la transacción actual
ROLLBACK;

```

```

-- 3. ¿Qué devolverá esta consulta? Justifique su respuesta.
SELECT*
FROM producto;

```

```

-- 4. Ejecutamos la siguiente transacción
STARTTRANSACTION;
INSERTINTO producto (id, nombre) VALUES (4, 'Cuarto');
SELECT*FROM producto;
ROLLBACK;

```

```
-- 5. ¿Qué devolverá esta consulta? Justifique su respuesta.
SELECT*FROM producto;
```

```
-- 6. Ejecutamos la siguiente transacción
INSERTINTO producto (id, nombre) VALUES (5, 'Quinto');
ROLLBACK;
```

```
-- 7. ¿Qué devolverá esta consulta? Justifique su respuesta.
SELECT*FROM producto;
```

```
-- 8. ¿Se puede hacer ROLLBACK de instrucciones de tipo DDL (CREATE,
-- ALTER,DROP, RENAME y TRUNCATE)?
```

DIFICULTAD BAJA

3.2.1.2 Responde las siguientes cuestiones:

1. ¿Qué son las propiedades *ACID*?
2. ¿Es posible realizar transacciones sobre tablas **MyISAM** de MySQL?
3. Considera que tenemos una tabla donde almacenamos información sobre cuentas bancarias definida de la siguiente manera:

```
CREATETABLE cuentas (
idINTEGER UNSIGNED PRIMARYKEY,
saldo DECIMAL(11,2) CHECK (saldo >=0)
);
```

Suponga que queremos realizar una transferencia de dinero entre dos cuentas bancarias con la siguiente transacción:

```
STARTTRANSACTION;
UPDATE cuentas SET saldo = saldo -100WHEREid=20;
UPDATE cuentas SET saldo = saldo +100WHEREid=30;
COMMIT;
```

- ✓ ¿Qué ocurriría si el sistema falla o si se pierde la conexión entre el cliente y el servidor después de realizar la primera sentencia UPDATE?
- ✓ ¿Qué ocurriría si no existiese alguna de las dos cuentas (id = 20 e id = 30)?

- ✓ ¿Qué ocurriría en el caso de que la primera sentencia UPDATE falle porque hay menos de 100 € en la cuenta y no se cumpla la restricción del CHECK establecida en la tabla?

DIFICULTAD BAJA

3.2.1.3 Crea una base de datos llamada `cine` que contenga dos tablas con las siguientes columnas.

Tabla `cuentas`:

- ✓ `id`: entero (clave primaria autoincrement).
- ✓ `nif`: cadena de 9 caracteres (clave alternativa).
- ✓ `saldo`: Real sin signo no nulo.
- Inserta tres cuentas.

Tabla `entradas`:

- ✓ `id`: entero (clave primaria).
- ✓ `butaca`: entero sin signo no nulo.
- ✓ `id_cuenta`: entero sin signo (clave ajena no nula que apunta a `cuentas.id`).
- Inserta tres entradas.

Una vez creada la base de datos, las tablas e insertadas las filas deberéis crear un procedimiento llamado `comprar_entrada` con las siguientes características:

- ✓ El procedimiento recibe 3 parámetros de **entrada** (`pidEntrada int`, `pButaca int`, `id_cuenta int`).
- ✓ Mostrará un mensaje de éxito si la compra de la entrada se ha podido realizar con éxito.
- ✓ Capturará el error en caso de intentar insertar una entrada cuya clave primaria ya existe: **ERROR 1062 (Duplicate entry for PRIMARY KEY)**. También revertirá la transacción que provocó el error.
- ✓ Capturará el error en caso de intentar insertar una entrada cuya clave ajena apunta a una cuenta inexistente: **ERROR 1452 (Cannot add or update a child row: a foreign key constraint fails)**. También revertirá la transacción que provocó el error.
- ✓ Capturará el error en caso de intentar insertar una entrada con un valor que está fuera de rango: **ERROR 1264 (Out of range value)**. Ejemplo: Insertar una entrada en la butaca -5. También revertirá la transacción que provocó el error.
- ✓ En caso de error el procedimiento siempre mostrará los valores de `MYSQL_ERRNO`, `RETURNED_SQLSTATE`, `MESSAGE_TEXT`.
- ✓ **Crea un solo manejador para todos los errores citados.**
- ✓ El procedimiento de compra realiza los siguientes pasos:

1. Inicia una transacción.
2. Actualiza la columna `saldo` de la tabla `cuentas` cobrando 5 euros a la cuenta con el `id_cuenta` adecuada.
3. Inserta una fila en la tabla `entradas` indicando la butaca (`id_butaca`) que acaba de comprar el usuario (`nif`).
4. Finaliza la transacción.
5. Comprueba si ha ocurrido algún error en las operaciones anteriores. Si no ocurre ningún error entonces aplica un `COMMIT` a la transacción y si ha ocurrido algún error aplica un `ROLLBACK`.

```
-- Infracción de clave primaria, ya existe ese ID en entradas
-- ¡Se debe revertir el decremento de saldo!
call comprar_entrada (2, 46, 3);
```

Filter Rows: | Export: | Wrap Cell Content:

Mensaje de error

Nº error: 1062
 Código de error: 23000
 Mensaje: Duplicate entry '2' for key 'entradas.PRIMARY'
 La transacción se ha revertido.

```
-- Infracción de rango de valor de campo, butaca no admite enteros negativos
-- ¡Se debe revertir el decremento de saldo!
call comprar_entrada (4, -5, 3);
```

Filter Rows: | Export: | Wrap Cell Content:

Mensaje de error

Nº error: 1264
 Código de error: 22003
 Mensaje: Out of range value for column 'butaca' at row 1
 La transacción se ha revertido.

```
-- Infracción de clave ajena, no existe ese ID de cuenta
-- También se revertirá la transacción pero no se habrá decrementado
-- el saldo de la cuenta al no existir la cuenta
call comprar_entrada (5, 46, 4);
```

Filter Rows: | Export: | Wrap Cell Content:

Mensaje de error

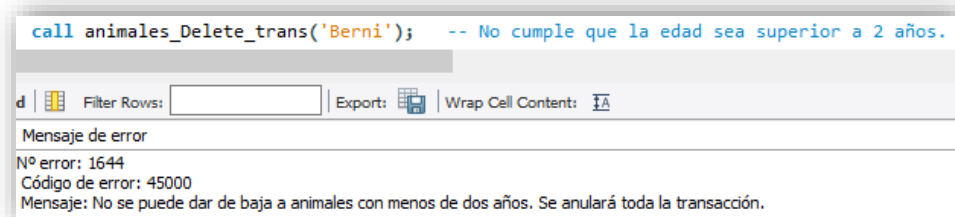
Nº error: 1452
 Código de error: 23000
 Mensaje: Cannot add or update a child row: a foreign key constraint fails ('test'. 'entradas', CONSTRAINT 'ent'
 La transacción se ha revertido.

DIFICULTAD MEDIA

3.2.2 BD Circo

3.2.2.1 Crear un procedimiento llamado “*animales_Delete_trans*” que amplíe el procedimiento “*animales_Delete*” del apartado sección Control de Errores.

- ✓ En el ejercicio original las excepciones lanzadas en caso de errores no se capturaban, sino que se lanzaban al sistema que detenía la ejecución.
- ✓ En esta ocasión vamos a capturar todas las excepciones lanzadas. El manejador de todas estas excepciones lo que hará será mostrar por pantalla el error con un SELECT y después haz un ROLLBACK para revertirla.
- ✓ En el ejercicio original el borrado de ANIMALES_ARTISTAS y de ANIMALES se hacía uno detrás del otro.
- ✓ En esta ocasión el borrado de ANIMALES_ARTISTAS se hará antes de comprobar la edad del animal. Pero ambos borrados pertenecerán a la misma transacción, si algo falla se revertirá la transacción.
- Para comprobar que lo has hecho correctamente, asegúrate que, después de eliminar las filas ANIMALES_ARTISTAS, se produce una condición de error y lanzas un SIGNAL. Capturarás ese error, informarás de él por pantalla y revertirás la transacción.
- También puedes comprobar cómo si consultas la tabla antes del ROLLBACK los datos han desaparecido y después del ROLLBACK los datos son repuestos.
- Ejemplo:



DIFICULTAD BAJA

3.2.2.2 Estamos reestructurando las gradas de las pistas (tabla PISTAS). Queremos contar en nuestro sistema con un procedimiento llamado “*mueveAforo*” que permita mover un número de localidades de una pista a otra. El procedimiento recibirá como parámetros el nombre de la pista origen, el de la de destino y el aforo a mover. Se trata de restar el aforo a la pista de origen para sumárselo a la de destino.

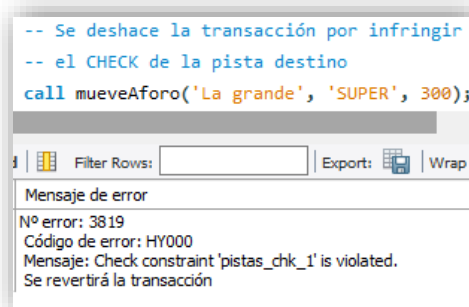
- ✓ Ojo, si algo falla no debe ejecutarse ninguna de las dos modificaciones. Son atómicas, o se ejecutan las dos o no se ejecuta ninguna.
- ✓ Tras comprobar la existencia de la pista de origen réstale el aforo, luego comprueba la existencia de la pista destino y súmale el aforo. En caso de no existir una pista se lanzará una excepción.
- ✓ Recuerda que el campo aforo de PISTAS está sujeta una restricción CHECK (aforo<=1000 AND aforo >=0).

- ✓ No compruebes nada relacionado con el aforo, que sea el sistema el que lance esa excepción.
- ✓ Usa un solo manejador de acción EXIT y de error SQLEXCEPTION.



nombre	aforo
La grande	350
LATERAL 1	300
LATERAL 2	550
SUPER	1000

Se ha revertido la resta de aforo en 'La grande'



nombre	aforo
La grande	350
LATERAL 1	300
LATERAL 2	550
SUPER	1000

Se ha revertido la resta de aforo en 'La grande'

DIFICULTAD BAJA

3.2.3 BD Formación

3.2.3.1 Crea un procedimiento de nombre “trasvaseNota”, que reciba tres parámetros: el código de un curso, el código del alumno1 y el código del alumno2. El procedimiento restará 0.5 puntos la nota del primer alumno en el curso, y le subirá 0.5 puntos la nota al segundo alumno en dicho curso.

- ✓ Obviamente, si alguno de los dos alumnos no existe o el curso no existe, etc. no se debe actualizar la nota del otro alumno. Se debe comprobar cualquier error y lanzar un SIGNAL.
- ✓ También debemos controlar que las notas se queden en el rango permitido (entre 0 y 10). Si no es así no se modificará ninguna nota.
- ✓ Si deseas “provocar” un error (cuando por ejemplo un alumno no esté matriculado en un curso), usa la sentencia SIGNAL con SQLSTATE ‘45000’ y, capturándola con un HANDLER, realiza un ROLLBACK.
- ✓ Mostrar los mensajes pertinentes tanto si se ha realizado correctamente como si ha habido algún error.
- ✓ Si no hay errores recuerda finalizar la transacción con un COMMIT.
- ✓ **NOTA:** Para que tenga sentido usar transacciones en este ejercicio lo mejor es hacer las comprobaciones solo para el primer alumno, actualizar su nota y después hacer lo mismo con el segundo.

1. Abrimos la transacción

2. Comprobamos existencia del curso.
3. Comprobamos existencia del **alumno 1**, que se halla matriculado al curso y que tiene una nota no nula mayor o igual que 0,5.
4. Si el **alumno 1** cumple las condiciones entonces hacemos el UPDATE al alumno1 sino lanzamos error con SIGNAL y la capturamos con CATCH. Desharemos también la transacción, aunque en este caso no había más instrucciones en ella.
5. Comprobamos existencia del **alumno2**, que se halla matriculado al curso y que tiene una nota no nula menor o igual que 9,5.
6. Si cumple las condiciones entonces hacemos el UPDATE al **alumno 2** con un **COMMIT** sino lanzamos error con SIGNAL y **ROLLBACK** en el CATCH para que deshaga la transacción (primer UPDATE).

```
-- El alumno 2 no existe (hay que deshacer el decremento a A1)
CALL trasvaseNota ('C1','A1','B5');
```

id | Filter Rows: | Export: | Wrap Cell Content: |

Mensaje de error

Nº error: 1644
Código de error: 45000
Mensaje: Error: el alumno 2 no existe.
La transacción se ha revertido.

```
-- El alumno 2 no puede tener más de 10
CALL trasvaseNota ('C1','A1','A3');
select * from alumnoscurtillos;
CALL trasvaseNota ('C1','A1','A3');
select * from alumnoscurtillos;
CALL trasvaseNota ('C1','A1','A3');
select * from alumnoscurtillos;
```

Mensaje
Trasvase realizado OK

CODAL	CODCUR	NOTA
A1	C1	8.5
A2	C2	8.0
A3	C1	9.5
A3	C2	9.0
NULL	NULL	NULL

Mensaje
Trasvase realizado OK

CODAL	CODCUR	NOTA
A1	C1	8.0
A2	C2	8.0
A3	C1	10.0
A3	C2	9.0
NULL	NULL	NULL

Mensaje de error
Nº error: 1644
Código de error: 45000
Mensaje: Error: El alumno 2 no puede tener más de 10.
La transacción se ha revertido.

CODAL	CODCUR	NOTA
A1	C1	8.0
A2	C2	8.0
A3	C1	10.0
A3	C2	9.0
NULL	NULL	NULL

DIFICULTAD BAJA

3.2.3.2 Crea un procedimiento de nombre “*convalidacion*”, que reciba dos parámetros: el código de un curso de origen y el curso convalidable. Es decir, convalidaremos a los alumnos el curso convalidable que tengan aprobado el curso de origen. La convalidación se realizará matriculando al alumno en el curso convalidable con la misma nota que tiene en el curso de origen **siempre que esté aprobado**. Nos hallaremos ante las siguientes restricciones:

- ✓ Las convalidaciones de los alumnos que han aprobado el curso de origen deben realizarse en bloque, es decir, **o se realizan todas o ninguna**.
- ✓ Cualquier error que surgiera provocará que se reviertan todas las inserciones en ALUMNOSCURSILLOS anteriores al error (convalidaciones ya realizadas).
- ✓ Errores:
 - Que la nota de un alumno matriculado al curso de origen sea nula.
 - Una violación de la clave primaria porque el alumno se hallaba ya matriculado en el curso convalidable. (**ERROR 1062**)
 - Si se ejecuta correctamente informaremos al usuario del número de convalidaciones realizadas.
 - En caso de errores informaremos de error y, si procede, del número de convalidaciones revertidas.
- ✓ Usa transacciones, SIGNAL y HANDLER.
- ✓ CLOSE cCursor; se debe ejecutar al final del HANDLER (y no al principio).

Convalidaciones correctas

1

```
SELECT * FROM alumnoscurtillos;
```

CODAL	CODCUR	NOTA
A1	C1	9.0
A2	C2	8.0
A3	C1	9.0
A3	C2	9.0

2



CODAL	CODCUR	NOTA
A1	C1	9.0
A1	C3	9.0
A2	C2	8.0
A3	C1	9.0
A3	C2	9.0
A3	C3	9.0

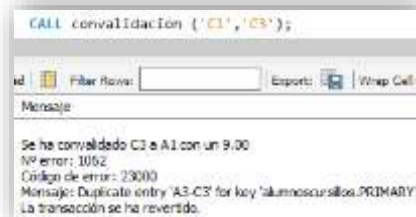
Convalidaciones fallidas (infracción de PK)

1

```
SELECT * FROM alumnoscurtillos;
```

CODAL	CODCUR	NOTA
A1	C1	9.0
A2	C2	8.0
A3	C1	9.0
A3	C2	9.0
A3	C3	9.0

2



CODAL	CODCUR	NOTA
A1	C1	9.0
A2	C2	8.0
A3	C1	9.0
A3	C2	9.0
A3	C3	9.0

(Sin cambios)

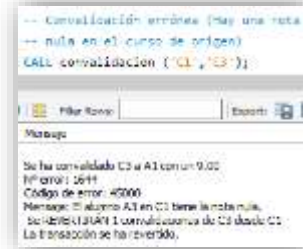
Convalidaciones fallidas (nota nula)

1

```
SELECT * FROM alumnoscurtillos;
```

CODAL	CODCUR	NOTA
A1	C1	9.0
A2	C2	8.0
A3	C1	NULL
A3	C2	9.0

2



CODAL	CODCUR	NOTA
A1	C1	9.0
A2	C2	8.0
A3	C1	NULL
A3	C2	9.0

(Sin cambios)

DIFICULTAD MEDIA

3.3 Transacciones II

3.3.1 Sin base de datos

3.3.1.1 Responde las siguientes cuestiones:

1. ¿Cuáles son los tres problemas de concurrencia en el acceso a datos que pueden suceder cuando se realizan transacciones? Ponga un ejemplo para cada uno de ellos.
2. Cuando se trabaja con transacciones, el SGBD puede bloquear conjuntos de datos para evitar o permitir que sucedan los problemas de concurrencia comentados en el ejercicio anterior. ¿Cuáles son los cuatro niveles de aislamiento que se pueden solicitar al SGBD?
3. ¿Cuál es el nivel de aislamiento que se usa por defecto en las tablas **InnoDB** de MySQL?

DIFICULTAD BAJA

3.4 Triggers

3.4.1 BD Formación

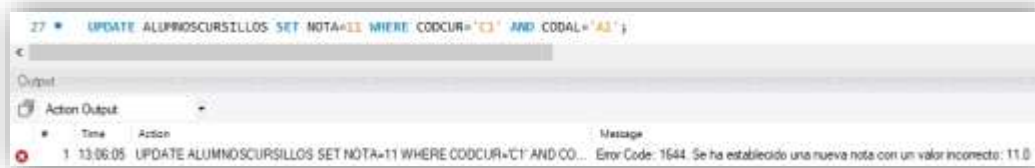
- Implementa un disparador llamado "tr_modificacion_nota" sobre la base de datos Formación que impida que la nota de un alumno sea superior a 10 o inferior a 0. Los disparadores "vigilarán" las modificaciones. En caso de que se produzca un intento de este tipo, lanzaremos un SIGNAL de error, e insertaremos un registro en la tabla llamada "CONTROL" de campos (usuario, instrucción, fecha_hora, alumno, curso, nota_previa, nota_intento).
- Tabla CONTROL donde almacenaremos información sobre el intento que infringe el intervalo posible de la nota.

```

CREATE TABLE CONTROL
(USUARIO VARCHAR(128) NOT NULL,
FECHA_HORA DATETIME NOT NULL,
CODAL VARCHAR(2) NOT NULL,
CODCUR VARCHAR(2) NOT NULL,
NOTA_PREVIA decimal(3,1) NOT NULL,
NOTA_INTENTO decimal(3,1) NOT NULL) ENGINE=MyISAM;

```

- Recuerda que para que SIGNAL no revierta la inserción en “CONTROL” la tabla debe contar con ENGINE=MyISAM en su definición.
- El usuario se obtiene con CURRENT_USER().



SELECT * FROM bdformacion.control;

id	USUARIO	FECHA_HORA	CODAL	CODCUR	NOTA_PREVIA	NOTA_INTENTO
1	root@localhost	2021-05-04 13:05:20	A1	C1	9.0	11.0

DIFICULTAD BAJA

- 3.4.1.1** Implementa un trigger “tr_modificacion_nota2” llamado que amplíe el trigger “tr_modificacion_nota” para que, en vez de lanzar un error, inserte la nota, pero modificando el nuevo valor a 10 si se intenta con uno mayor de 10 y 0 si se intenta con uno menor que 0.

DIFICULTAD BAJA

3.4.2 BD Circo

- 3.4.2.1** Crea unos triggers llamados “tr_ganacias_atracciones_insert”, “tr_ganacias_atracciones_update” y “tr_ganacias_atracciones_delete” que comprueben que se cumplen las siguientes restricciones:
1. El campo ganancias de la tabla ATRACCIONES se actualice cuando se añadan, borren o modifiquen datos en la tabla ATRACCION_DIA.
 2. Si al añadir una celebración nueva (ATRACCION_DIA) la fecha_inicio en ATRACCIONES es NULL, se debe de actualizar con la fecha actual.
- ✓ Nota: Debéis de tener en cuenta que tanto la fecha como las ganancias en ATRACCIONES pueden tener valores nulos. Dad un valor a esas columnas cuando

sean null (recordar que si operamos matemáticamente o concatenamos una columna con valor null siempre devolverá null).

DIFICULTAD BAJA

- 3.4.2.2** Modifica la tabla `ATRACCIONES` y añade una nueva columna de nombre contador, de tipo numérico y valor por defecto de cero, que lleve cuenta de cuantas veces se ha celebrado la atracción y que se actualice con cualquier operación sobre la tabla `ATRACCION_DIA`.

Crea los triggers necesarios llamados “tr_contador_atracciones_insert” y “tr_contador_atracciones_delete”.

- ✓ **Nota:** Los datos ya añadidos tendrán como valor null. Realiza una operación de UPDATE para ponerlos a su valor correcto.

DIFICULTAD BAJA

- 3.4.2.3** Crea unos triggers llamados “tr_pista_aforo_insert” y “tr_pista_aforo_update” que impidan añadir o modificar una pista que dé como resultado un aforo superior a 1000 o inferior a 10 (esto lo podríamos implementar con un CHECK, pero vamos a practicar el uso de triggers).

DIFICULTAD BAJA

- 3.4.2.4** Crea un trigger llamado “tr_pista_aforo_insert” que si se intenta dar de alta un nuevo artista y se envía un nif_jefe que no exista, se cambie su valor por null.

DIFICULTAD BAJA

- 3.4.2.5** Crea una tabla de nombre REGISTRO con las columnas:

- ✓ id autonumérica Clave primaria
- ✓ usuario: varchar(100)
- ✓ tabla: varchar(100)
- ✓ operacion: varchar(10)
- ✓ datos_antiguos: varchar(100) (guardarán los datos nombre_pista:aforo que se borren o modifiquen)
- ✓ datos_nuevos: varchar(100) (guardarán los datos nombre_pista:aforo que se añadan o modifiquen)
- ✓ fecha-hora: datetime (el valor por defecto será la fecha-hora del sistema)

```

CREATETABLE `REGISTRO` (
  `id` int NOTNULL AUTO_INCREMENT,
  `usuario` varchar(100) NOTNULL,
  `tabla` varchar(100) NOTNULL,
  `operacion` varchar(10) NOTNULL,
  `fecha-hora` datetime NOTNULL DEFAULT CURRENT_TIMESTAMP,
  `datos_antiguos` varchar(100) DEFAULT NULL,
  `datos_nuevos` varchar(100) DEFAULT NULL,
  PRIMARYKEY (`id`)
);

```

Haz que se registren en la tabla REGISTRO las operaciones de alta, baja y modificación sobre la tabla PISTAS por medio de los triggers

“pistas_addRegistro_INSERT”, “pistas_addRegistro_DELETE” y “pistas_addRegistro_UPDATE”.

Por ejemplo:

- ✓ 'angel' - 'PISTAS' - 'ALTA' - null - 'pista_nueva:1000' - '01-01-2000 17:00:00'
(el campo *datos_antiguos* es nulo ya que estamos a dar de alta una nueva pista).
- ✓ 'luis' - 'PISTAS' - 'BAJA' - 'pista_borrar:1000' - null - '01-01-2000 18:00:00'
(el campo *datos_nuevos* es nulo ya que estamos a dar de baja una pista).
- ✓ 'pepe' - 'PISTAS' - 'MODIFICAR' - 'pista_modificar:1000' - 'pista_modificar:1500' - '01-01-2000 19:00:00'
(el campo *datos_nuevos* guarda los datos modificados y el campo *datos_antiguos* guarda los datos antes de la modificación).

```

INSERT INTO `PISTAS` (`nombre`, `aforo`) VALUES ('LATERAL3', '120');
UPDATE `PISTAS` SET `aforo` = '150' WHERE (`nombre` = 'LATERAL3');
DELETE FROM `PISTAS` WHERE (`nombre` = 'LATERAL3');
SELECT * FROM REGISTRO;

```

id	usuario	tabla	operacion	fecha-hora	datos_antiguos	datos_nuevos
1	root@localhost	PISTAS	ALTA	2021-05-06 22:22:37	NULL	LATERAL3:120
2	root@localhost	PISTAS	MODIFICAR	2021-05-06 22:22:37	LATERAL3:120	LATERAL3:150
3	root@localhost	PISTAS	BAJA	2021-05-06 22:22:37	LATERAL3:150	NULL

DIFICULTAD BAJA

3.4.2.6 Crea una tabla de nombre CONTADOR con las columnas:

- ✓ id autonumérica Clave primaria
- ✓ tipo: varchar(100) no nulo
- ✓ valor: int no nulo

```

CREATE TABLE `CONTADOR` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `tipo` VARCHAR(100) NOT NULL,
  `valor` INT NOT NULL,
  PRIMARY KEY (`id`));

```

Añade dos filas con los valores siguientes:

id	tipo	valor
1	pistas	0
2	animales	0

Actualizamos los datos con el siguiente código:

```

UPDATE CONTADOR
SET valor = (SELECT COUNT(*) FROM PISTAS)
WHERE tipo = 'pistas';

UPDATE CONTADOR
SET valor = (SELECT COUNT(*) FROM ANIMALES)
WHERE tipo = 'animales';

```

	id	tipo	valor
▶	1	pistas	4
	2	animales	9

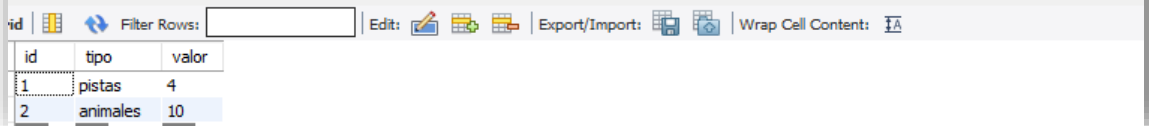
Haz que cada vez que haya alguna operación que modifique (alta/baja) el número de pistas o de animales, se actualice el número total de los mismos mediante los triggers:

- ✓ animales_gestContador_INSERT
- ✓ animales_gestContador_DELETE
- ✓ pistas_gestContador_INSERT
- ✓ pistas_gestContador_DELETE


```

INSERT INTO `PISTAS` (`nombre`, `aforo`) VALUES ('ESPECTACULAR', '250');
DELETE FROM `PISTAS` WHERE (`nombre` = 'ESPECTACULAR');
INSERT INTO `ANIMALES` (`nombre`, `tipo`, `años`, `peso`, `estatura`, `nombre_atraccion`, `nombre_pista`)
VALUES ('Jaim', 'Mono', '1', '15', '.95', 'El gran carnívoro', 'SUPER');

```



id	tipo	valor
1	pistas	4
2	animales	10

DIFICULTAD BAJA

- 3.4.2.7** Haz que no se pueda añadir un nuevo animal si el tipo es 'León' y el número de años es mayor que 20 mediante un trigger llamado “tr_add_animal”.
- En caso de no cumplirse la condición lanzará una excepción que capturará el sistema (no nosotros).
- Nota:** Fijarse que, si el alta la realizáramos a través de un procedimiento almacenado, la comprobación ya puede ir allí y no necesitaríamos el uso de triggers. Como queremos prevenir que se realice “a pelo” como instrucción individual, se hará por medio de triggers.

```

26 INSERT INTO `ANIMALES` (`nombre`, `tipo`, `años`, `peso`, `estatura`, `nombre_atraccion`, `nombre_pista`)
27 VALUES ('El cosechobres', 'León', 25, 120, 1.2, 'El gran felino', 'LATERALI');

```



#	Time	Action	Message
1	12:26:57	INSERT INTO 'ANIMALES' ('nombre', 'tipo', 'años', 'peso', 'estatura', 'nombre_atrac...	Error Code: 1644. El tipo león no puede tener más de 20 años

DIFICULTAD BAJA

- 3.4.2.8** Crea un trigger llamado “tr_add_animal2” que, cuando se añada un nuevo animal, haga que dicho animal esté cuidado por el artista que cuida a menos animales.
- ✓ Deberás de tener en cuenta el caso en el que no haya ningún artista cuidando animales. En ese caso debes coger el primer artista que no sea jefe.
 - ✓ Fijarse que este trigger va a suponer que se añaden dos filas: una fila a la tabla ANIMALES y otra a la tabla ANIMALES_ARTISTAS. Si el INSERT INTO ANIMALES_ARTISTAS fallara, se revertiría el trigger y la instrucción que lo disparó (y otras que pertenecieran a la misma transacción).

Caso 1

nombre	tipo	anhos	peso	estatura	nombre_atraccion	nombre_pista
Berni	León	1	94	13.9	El gran carnívoro	LATERAL2
Caiman	Cocodrilo	1	71	14.1	El gran carnívoro	LATERAL2
Frank	Jirafa	3	146	15.14	Las jirafas	SUPER
Jazinto	Mono	13	NULL	NULL	NULL	NULL
Leo	León	3	121	14.1	El gran felino	SUPER
Lucas	Cocodrilo	2	120.34	NULL	NULL	LATERAL2
Peter	Mono	1	30	10.7	NULL	NULL
Princesa1	Jirafa	2	100	12.2	Las jirafas	LATERAL1
Princesa2	Jirafa	3	110	12.3	Las jirafas	LATERAL1

nombre_animal	nif_artista
Caiman	11111111A
Jazinto	11111111A
Leo	11111111A
Princesa1	11111111A
Caiman	22222222B
Princesa2	22222222B
Leo	44444444D
Berni	71111111A
Frank	71111111A
Lucas	71111111A
Peter	71111111A

Tablas originales

```
-- Si ejecutamos este código debería asignar el animal al artista con nif 22222222B.
INSERT INTO `ANIMALES` (`nombre`,`tipo`,`anhos`,`peso`,`estatura`,`nombre_atraccion`,`nombre_pista`)
VALUES ('El comehombres','León',2,120,1.2,'El gran felino','LATERAL1');
```

Inserción

nombre	tipo	anhos	peso	estatura	nombre_atraccion	nombre_pista
Berni	León	1	94	13.9	El gran carnívoro	LATERAL2
Caiman	Cocodrilo	1	71	14.1	El gran carnívoro	LATERAL2
El comehombres	León	2	120	1.2	El gran felino	LATERAL1
Frank	Jirafa	3	146	15.14	Las jirafas	SUPER
Jazinto	Mono	13	NULL	NULL	NULL	NULL
Leo	León	3	121	14.1	El gran felino	SUPER
Lucas	Cocodrilo	2	120.34	NULL	NULL	LATERAL2
Peter	Mono	1	30	10.7	NULL	NULL
Princesa1	Jirafa	2	100	12.2	Las jirafas	LATERAL1
Princesa2	Jirafa	3	110	12.3	Las jirafas	LATERAL1

nombre_animal	nif_artista
Caiman	11111111A
Jazinto	11111111A
Leo	11111111A
Princesa1	11111111A
Caiman	22222222B
Princesa2	22222222B
El comehombres	44444444D
Leo	44444444D
Berni	71111111A
Frank	71111111A
Lucas	71111111A
Peter	71111111A

Tablas modificadas

Caso 2

nombre	tipo	anhos	peso	estatura	nombre_atraccion	nombre_pista
NULL	NULL	NULL	NULL	NULL	NULL	NULL

nombre_animal	nif_artista
NULL	NULL

Tablas originales

```
-- Si ejecutamos este código debería asignar el animal al artista con nif 22222222B.
INSERT INTO `ANIMALES` (`nombre`,`tipo`,`anhos`,`peso`,`estatura`,`nombre_atraccion`,`nombre_pista`)
VALUES ('El comehombres','León',2,120,1.2,'El gran felino','LATERAL1');
```

Inserción

2 12:45:59 INSERT INTO `ANIMALES` (`nombre`,`tipo`,`anhos`,`peso`,`estatura`,`nombre_atrac... Error Code: 1644. No hay artistas para cuidar a animales

Error

DIFICULTAD MEDIA

3.4.3 BD Empresa (Falta solución)

3.4.3.1 Borra si existe y crea una tabla llamada “logInsercionesEmp” con los siguientes campos:

- ✓ Fecha y hora, usuario, id y nombre del empleado insertado, resultado de la inserción (“Error” o “OK”).

Crea un trigger llamado “tr_PKempleado” que compruebe si existe el empleado (la clave primaria) y no permita la inserción.

- Recuerda que para que SIGNAL no revierta la inserción en “logInsercionesEmp” la tabla debe contar con ENGINE=MyISAM en su definición.

DIFICULTAD BAJA

- 3.4.3.2** Crea un disparador llamado “tr_fechaempleado” para controlar la fecha de alta de los empleados: al intentar insertar uno con fecha posterior a la actual, se genera un error con un mensaje informativo por consola cancelando la inserción usando SIGNAL.

DIFICULTAD BAJA

- 3.4.3.3** Diseña un trigger llamado “tr_salarioHorario” que no deje modificar los salarios fuera del horario laboral. Solo dejará modificar de 9 a 2 y de 4 a 7 y solo en días laborables y excluirá también agosto entero. En caso de que no permita la modificación, mostrará un mensaje mostrando información sobre el error usando SIGNAL.

DIFICULTAD BAJA

- 3.4.3.4** Realiza una versión del ejercicio anterior llamado “tr_salarioHorario” que además guarde en una tabla “logActualizacionesSalarios” con campos fecha y hora, usuario, el id, nombre, anterior salario y nuevo salario del empleado insertado, y el resultado de la inserción (“Error” o “OK”).

- Recuerda que para que SIGNAL no revierta la inserción en “logActualizacionesSalarios” la tabla debe contar con ENGINE=MyISAM en su definición.

DIFICULTAD MEDIA