EJERCICIO TEMA 6

MENEJO DE ERRORES, TRANSACCIONES, CURSORES Y TRIGGERS

Crea una tabla ***alumno*** con las siguientes columnas:

id: entero sin signo (clave primaria).

nombre: cadena de 50 caracteres.

apellido1: cadena de 50 caracteres.

apellido2: cadena de 50 caracteres.

Ejercicio 1:

Crea un procedimiento llamado insertarAlumno que reciba los cuatro datos como parámetros de entrada y los inserte en la tabla.

- El procedimiento devolverá como salida un parámetro llamado error que tendrá un valor igual a 0 si la operación se ha podido realizar con éxito o 1 en caso contrario.
- Deberá manejar los errores que puedan ocurrir cuando se intenta insertar una fila que contiene una clave primaria repetida.

ERROR NO ENCONTRADO

A partir de la base de datos ***jardinería***:

Ejercicio 2:

Escribe un procedimiento (nombreClienteIf) que muestre el nombre de un cliente dado su código.

En caso de que no se encuentre, devolverá *"Cliente no encontrado"*.

- Utiliza ***if*** para controlar el si lo ha encontrado o no.

Ejercicio 3:

Vuelve a hacer el ejercicio anterior (nombreClienteHandler) pero ahora utiliza el manejador de errores (sin usar if)

- La consulta debe guardar el resultado en una variable mediante ***SELECT … INTO***

SQLException

Ejercicio 3:

Escribe un procedimiento (creaOficina) que cree una oficina a partir de un código, ciudad, pais, codigopostal, telefono y dirección.

- Si el codigo de la oficina ya existe, modificará los datos de dicha oficina con los recibidos como parámetros.

TRANSACCIONES

Ejercicio extraído de https://josejuansanchez.org/bd/unidad-11-teoria/index.html#ejercicios-pr%C3%A1cticos-de-transacciones

Ejercicio 1:

Ejecuta el siguiente *script* y resuelve las cuestiones que se plantean en cada caso

```
SET AUTOCOMMIT = 0;
SELECT @@AUTOCOMMIT;

DROP DATABASE IF EXISTS test;
CREATE DATABASE test CHARACTER SET utf8mb4;
USE test;

CREATE TABLE producto (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  precio DOUBLE
);

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


-- 1. Comprueba que las filas se han insertado en la tabla de forma correcta.
SELECT *
FROM producto;
```

Ahora vamos a simular que se ha perdido la conexión con el servidor antes que la transacción sea completada, observa que AUTOCOMMIT = 0. En Dbeaver tenemos los botones de conectar y desconectar en la barra de herramientas, si estás conectado desde la consola del servidor solo tienes que hacer EXIT.

Volvamos a conectarnos al servidor y ejecutamos las siguientes instrucciones:

```
USE test;
-- ¿Qué devolverá esta consulta?
SELECT *
FROM producto;
```

Ejercicio 2:

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

```
SET AUTOCOMMIT = 1;
SELECT @@AUTOCOMMIT;

DROP DATABASE IF EXISTS test;
CREATE DATABASE test CHARACTER SET utf8mb4;
USE test;

CREATE TABLE producto (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  precio DOUBLE
);

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


-- 1. ¿Qué devolverá esta consulta?
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
START TRANSACTION;
INSERT INTO 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
INSERT INTO producto (id, nombre) VALUES (5, 'Quinto');
ROLLBACK;

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

```sql
-- 8. Desactivamos el modo AUTOCOMMIT y borramos el contenido de la tabla
SET AUTOCOMMIT = 0;
SELECT @@AUTOCOMMIT;

DELETE FROM producto WHERE id > 0;

-- 9. Comprobamos que la tabla esta vacia
SELECT * FROM producto;

-- 10. Insertamos dos filas nuevas
INSERT INTO producto (id, nombre) VALUES (6, 'Sexto');
INSERT INTO producto (id, nombre) VALUES (7, 'Séptimo');
SELECT * FROM producto;

-- 11. Hacemos un ROLLBACK
ROLLBACK;

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

-- 13. Ejecutamos la siguiente transacción
SET AUTOCOMMIT = 0;
START TRANSACTION;
CREATE TABLE fabricante (id INT UNSIGNED);
INSERT INTO fabricante (id) VALUES (1);
SELECT * FROM fabricante;
ROLLBACK;

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

Más info en https://mariadb.com/kb/en/sql-statements-that-cause-an-implicit-commit/

PROCEDIMIENTO TRANSACCIONAL

Crea un base de datos llamada ***cine*** con las siguientes tablas:

- Tabla **cuentas:**
  - idCuenta: entero sin signo (pk).
  - saldo: real sin signo.
- Tabla **entradas:**
  - idButaca: entero sin signo (pk).
  - nif: cadena de 9 caracteres.

Ejercicio 1:

Crear un procedimiento llamado comprarEntrada que recibe 3 parámetros de entrada (nif, idCuenta , idButaca) y devolverá como salida un parámetro llamado **error** que tendrá un valor igual a **0** si la operación se ha podido realizar con éxito o **1** en caso contrario.

Pasos del procedimiento de compra:

1. Inicia una transacción.
2. Actualiza cuentas.saldo cobrando 5 euros a la cuenta con el idCuenta adecuado.
3. Inserta una fila en la tabla entradas indicando la butaca (idButaca) que acaba de comprar el usuario (nif).
4. Comprueba si ha ocurrido algún error en las operaciones anteriores. Si todo va bien aplica un COMMIT a la transacción y si ha ocurrido algún error haz ROLLBACK.
5. Debe manejar los errores ERROR 1264 (Out of range value) y ERROR 1062 (Duplicate entry for PRIMARY KEY).

¿Qué ocurre si se compra una entrada y le pasamos como parámetro un número de cuenta inexistente? ¿Ocurre algún error o podemos comprar la entrada? En caso de que exista algún error, ¿cómo podríamos resolverlo?.

CURSORES

Ejercicio 1:

Crea la base de datos **cursores** con una tabla llamada ***alumnos*** y 4 sentencias de inserción para inicializar la tabla que contiene las siguientes columnas:

- id (entero sin signo y clave primaria)
- nombre (cadena de caracteres)
- apellido1 (cadena de caracteres)
- apellido2 (cadena de caracteres)
- fechaNacimiento (fecha)

Tras crear la tabla se decide añadir una nueva columna llamada ***edad***. Escribe la sentencia SQL necesaria para modificar la tabla y añadir la nueva columna.

- Escribe una función llamada calcularEdad que reciba una fecha y devuelva el número de años que han pasado desde la fecha actual hasta la fecha pasada como parámetro.
- Escribe un procedimiento (actualizarColumnaEdad) que actualice la edad de todos los alumnos que ya existen en la tabla.
  - Utiliza un cursor para recorrer la tabla y modificar cada alumno.
  - Este procedimiento deberá utilizar la función calcularEdad.

EJERCICIO 2

Modifica la tabla **alumnos** del ejercicio anterior para añadir una columna **email.** Una vez hemos modificado la tabla necesitamos asignarle una cuenta de correo de forma automática a cada uno de ellos.

Crea una función (crearEmail) que a partir del nombre, apellido1, apellido2 y dominio, genere una dirección de email y la devuelva como salida. El formato del email de salida es el siguiente:

- El primer carácter del parámetro nombre.
- Los tres primeros caracteres del parámetro apellido1.
- Los tres primeros caracteres del parámetro apellido2.
- El carácter @.
- El dominio pasado como parámetro.

Ejemplo: crearEmail('Gonzalo', 'Ruis', 'Arenas', 'edu.gva.es') devolvería gruiare@edu.gva.es

Crea un procedimiento (actualizarColumnaEmail) que permita crear un email para todos los alumnos que ya existen en la tabla.

- Debes utilizar la función crearEmail

Crea un procedimiento (crearListaEmailsAlumnos) que devuelva la lista de emails de la tabla alumnos separados por un punto y coma.

- Ejemplo:lidia@ieslaencanta.es;roman@ieslaencanta.es;alex@ieslaencanta.es;elvira@ ieslaencanta.es;

TRIGGERS

Ejercicio 1:

Crea un trigger (TriggerCrearEmailBeforeInsert) sobre la tabla alumnos, para que si insertamos un nuevo registro en la tabla y el campo email es nulo se le asigne uno automáticamente.

- Debes utilizar la función crearEmail con el dominio 'noemail.com'.

Ejercicio 2:

Crea un trigger (triggerGuardarEmailAfterUpdate) sobre la tabla alumnos, para que cada vez que se modifica el email, inserte un nuevo registro en la tabla logCambiosEmail, cuyos campos son:

- id: clave primaria (entero autonumérico)
- idAlumno: id del alumno (entero)
- fechaHora: marca de tiempo con el instante del cambio (fecha y hora)
- oldEmail: valor anterior del email (cadena de caracteres)
- newEmail: nuevo valor con el que se ha actualizado

Ejercicio 3:

Crea un trigger (triggerGuardarAlumnosAfterDelete) sobre la tabla alumnos, para que cada vez que se elimine un alumno, inserte un nuevo registro en la tabla logAlumnosEliminados, cuyos campos son:

- id: clave primaria (entero autonumérico)
- idAlumno: id del alumno (entero)
- fechaHora: marca de tiempo
- nombre: nombre del alumno (cadena de caracteres)
- apellido1: 1er apellido (cadena de caracteres)
- apellido2: 2º apellido (cadena de caracteres)
- email: email del alumno (cadena de caracteres)