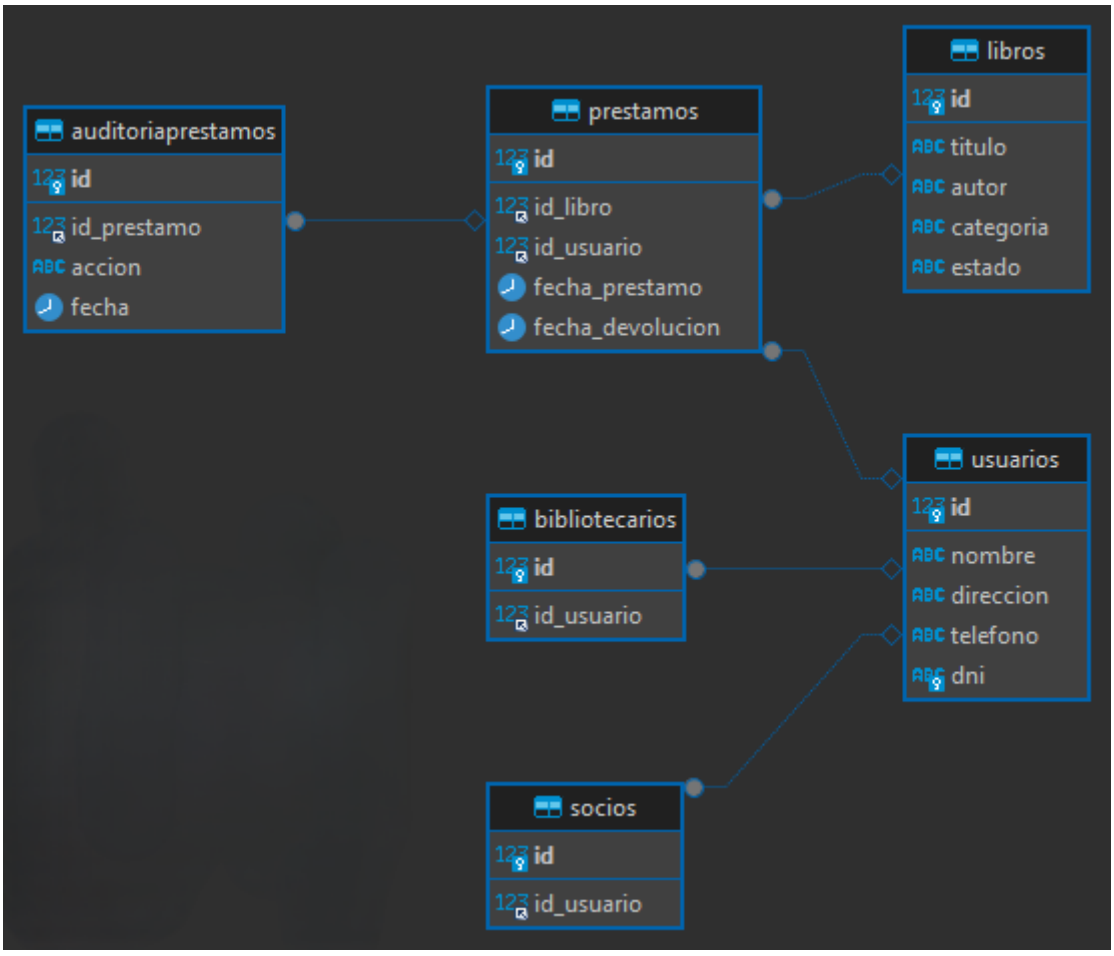
TP Evaluativo: “Base de datos Biblioteca”

Desarrolla el modelo de bases de datos para un sistema de gestión para una biblioteca. El sistema debe permitir el manejo de libros, autores, usuarios y préstamos. Además, deberás implementar varias funcionalidades avanzadas utilizando procedimientos almacenados (stored procedures), vistas (views), disparadores (triggers) y transacciones (transaction).
Para el desarrollo puedes tener en cuenta el siguiente esquema y modificarlo y adaptarlo a tus necesidades.

1. Diseño de la Base de Datos:
    - Crea una base de datos llamada Biblioteca.
    - Define las tablas con sus respectivos campos y relaciones según creas necesario. Mínimamente, debes considerar las siguientes entidades: Libros, Bibliotecario (proveedor del libro), Usuarios (lectores/clientes), Prestamos.
2. Procedimientos Almacenados (Stored Procedures):
    - Crea un procedimiento almacenado llamado registrar_prestamo que reciba como parámetros id_libro, id_usuario y fecha_prestamo. El procedimiento debe verificar si el libro ya está prestado (es decir, que no tiene una fecha de devolución) antes de registrar un nuevo préstamo.
3. Vistas:
    - Crea una vista llamada vista_prestamos_actuales que muestre los préstamos activos, incluyendo el nombre del libro, el nombre del usuario y la fecha de préstamo.
4. Disparadores (Triggers):
    - Crea un trigger llamado actualizar_fecha_devolucion que se ejecute antes de actualizar la tabla Prestamos y registre la fecha actual como fecha_devolucion cuando un préstamo se devuelve.
5. Transacciones:
    - Implementa una transacción que registre un nuevo préstamo, actualice el estado del libro a "prestado" y registre la operación en una tabla de auditoría llamada AuditoriaPrestamos con los campos id_auditoria, id_prestamo, accion, fecha.

Para comenzar con la actividad propuesta, pensamos las tablas y entidades que debían estar en la base de datos para poder después manipularla correctamente.
Utilizamos una base de datos relacional ya utilizamos tablas para mostrar la información , cada una de las cuales representa una entidad del mundo real. Dentro las tablas, cada tabla contiene columnas (campos) que representan los atributos de la entidad y filas (registros) que representan instancias de la entidad. A su vez, cada tabla tiene una columna o un conjunto de columnas que se utilizan como identificador único para cada fila que llamamos “clave primaria”. Ahora bien, las relaciones entre tablas se definen mediante claves foráneas, que son columnas que contienen valores que hacen referencia a la clave primaria de otra tabla. Esto permite mantener la integridad referencial.
Luego de pensar el diseño de tablas realizamos el diagrama entidad relación (también conocido como diagrama ER o diagrama ERD o simplemente ERD) que muestra cómo interactúan las entidades (personas, objetos y conceptos).



Utilizamos el lenguaje SQL para manipular y definir los datos y MySQL como sistema de gestión de bases de datos (DBMS, por sus siglas en inglés).

1. Diseño de la base de datos

Según el enunciado mínimamente, debíamos considerar las siguientes entidades: Libros, Bibliotecario (proveedor del libro), Usuarios (lectores/clientes), Prestamos.

El script quedo de esta forma:

- En primer lugar, creamos la base de datos Biblioteca y luego la llamamos para poder ejecutar los scripts.
-

```
CREATE DATABASE Biblioteca;
USE Biblioteca;
```

- Se realiza la creación de las tablas en la base de datos

```
CREATE TABLE Usuarios (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(255) NOT NULL,
    direccion VARCHAR(255) NOT NULL,
    telefono VARCHAR(20) NOT NULL,
    dni VARCHAR(20) UNIQUE NOT NULL
);

CREATE TABLE Libros (
    id INT PRIMARY KEY AUTO_INCREMENT,
    titulo VARCHAR(255) NOT NULL,
    autor VARCHAR(255) NOT NULL,
    categoria VARCHAR(255),
    estado ENUM('disponible', 'prestado') NOT NULL DEFAULT 'disponible'
);

CREATE TABLE Bibliotecarios (
    id INT PRIMARY KEY AUTO_INCREMENT,
    id_usuario INT,
    FOREIGN KEY (id_usuario) REFERENCES Usuarios(id)
);

CREATE TABLE Socios (
    id INT PRIMARY KEY AUTO_INCREMENT,
    id_usuario INT,
    FOREIGN KEY (id_usuario) REFERENCES Usuarios(id)
);
```

```
CREATE TABLE Prestamos (
    id INT PRIMARY KEY AUTO_INCREMENT,
    id_libro INT,
    id_usuario INT,
    fecha_prestamo DATETIME NOT NULL,
    fecha_devolucion DATETIME,
    FOREIGN KEY (id_libro) REFERENCES Libros(id),
    FOREIGN KEY (id_usuario) REFERENCES Usuarios(id)
);

CREATE TABLE AuditoriaPrestamos (
    id INT PRIMARY KEY AUTO_INCREMENT,
    id_prestamo INT,
    accion VARCHAR(255) NOT NULL,
    fecha DATETIME NOT NULL,
    FOREIGN KEY (id_prestamo) REFERENCES Prestamos(id)
);
```

- El paso que sigue es ingresar datos en las tablas, respetando los datos pedidos por cada tabla.

Para poder utilizar la base de datos Biblioteca debemos llamarla ya que estamos utilizando un nuevo script.

```
USE Biblioteca;
```

Luego comenzamos con las inserciones en las tablas.

```
INSERT INTO Usuarios (nombre, direccion, telefono, dni) VALUES
('Juan Pérez', 'Calle Falsa 123', '123456789', '12345678A'),
('María López', 'Avenida Siempre Viva 456', '987654321', '87654321B'),
('Carlos Sánchez', 'Calle del Sol 789', '112233445', '11223344C'),
('Ana Gómez', 'Boulevard de la Luna 101', '556677889', '22334455D'),
('Laura Rodríguez', 'Plaza Mayor 202', '998877665', '33445566E'),
('Miguel Fernández', 'Camino Real 303', '667788990', '44556677F');


INSERT INTO Libros (titulo, autor, categoria, estado) VALUES
('Cien Años de Soledad', 'Gabriel García Márquez', 'Novela', 'disponible'),
('Don Quijote de la Mancha', 'Miguel de Cervantes', 'Novela', 'disponible'),
('1984', 'George Orwell', 'Distopía', 'disponible'),
('El Principito', 'Antoine de Saint-Exupéry', 'Infantil', 'disponible'),
('Matar a un ruiseñor', 'Harper Lee', 'Novela', 'disponible'),
('La Odisea', 'Homero', 'Épica', 'disponible');


INSERT INTO Bibliotecarios (id_usuario) VALUES
(1),
(2),
(3),
(4),
(5),
(6);
```

```
INSERT INTO Bibliotecarios (id_usuario) VALUES
(1),
(2),
(3),
(4),
(5),
(6);

INSERT INTO Socios (id_usuario) VALUES
(1),
(2),
(3),
(4),
(5),
(6);


INSERT INTO Prestamos (id_libro, id_usuario, fecha_prestamo, fecha_devolucion) VALUES
(1, 1, '2024-06-01 10:00:00', '2024-06-15 10:00:00'),
(2, 2, '2024-06-02 11:00:00', '2024-06-16 11:00:00'),
(3, 3, '2024-06-03 12:00:00', '2024-06-17 12:00:00'),
(4, 4, '2024-06-04 13:00:00', '2024-06-18 13:00:00'),
(5, 5, '2024-06-05 14:00:00', '2024-06-19 14:00:00'),
(6, 6, '2024-06-06 15:00:00', '2024-06-20 15:00:00');
```

```
INSERT INTO AuditoriaPrestamos (id_prestamo, accion, fecha) VALUES
(1, 'Prestamo Creado', '2024-06-01 10:00:00'),
(2, 'Prestamo Creado', '2024-06-02 11:00:00'),
(3, 'Prestamo Creado', '2024-06-03 12:00:00'),
(4, 'Prestamo Creado', '2024-06-04 13:00:00'),
(5, 'Prestamo Creado', '2024-06-05 14:00:00'),
(6, 'Prestamo Creado', '2024-06-06 15:00:00');
```

2. Procedimientos Almacenados (Stored Procedures):

Sabemos que un procedimiento almacenado es un conjunto de instrucciones que se pueden llamar desde otras consultas o desde otros procedimientos almacenados. Un procedimiento puede tomar argumentos de entrada y mostrar valores como resultados. En nuestro caso se pedía que se creara un procedimiento almacenado llamado registrar_prestamo que reciba como parámetros id_libro, id_usuario y fecha_prestamo. El procedimiento debía verificar si el libro ya está prestado (es decir, que no tiene una fecha de devolución) antes de registrar un nuevo préstamo.

Como en el caso anterior para poder utilizar la base de datos Biblioteca debemos llamarla ya que estamos utilizando un nuevo script.

```
USE Biblioteca;
```

Para crear el procedimiento almacenado, usamos un bloque anónimo para evitar problemas de delimitador:

```
CREATE PROCEDURE registrar_prestamo(
    IN p_id_libro INT,
    IN p_id_usuario INT,
    IN p_fecha_prestamo DATETIME
)
BEGIN
```

Luego verificamos si el libro ya está prestado, si el libro está prestado, muestra error, y si el libro no está prestado, registra el nuevo préstamo.
El procedimiento completo queda de esta forma:

```sql
CREATE PROCEDURE registrar_prestamo(
    IN p_id_libro INT,
    IN p_id_usuario INT,
    IN p_fecha_prestamo DATETIME
)
BEGIN
    IF EXISTS (
        SELECT 1
        FROM Prestamos
        WHERE id_libro = p_id_libro
        AND fecha_devolucion IS NULL
    ) THEN

        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'El libro ya está prestado.';
    ELSE
        INSERT INTO Prestamos (id_libro, id_usuario, fecha_prestamo)
        VALUES (p_id_libro, p_id_usuario, p_fecha_prestamo);
    END IF;
END;
```

Para asegurarnos de que el procedimiento se ha creado correctamente, ejecutamos:

SHOW PROCEDURE STATUS WHERE Db = 'Biblioteca';

Y luego para probar el procedimiento, lo llamamos con:

CALL registrar_prestamo(1, 2, '2024-06-23 10:00:00');

3. Vistas:

Las vistas son objetos que contienen una consulta definida y devuelven resultados como una tabla virtual, siendo útiles para simplificar consultas complejas y asegurar consistencia en los resultados.
Para crear la vista vista_prestamos_actuales que muestre los préstamos activos junto con el nombre del libro, el nombre del usuario y la fecha de préstamos utilizamos las tablas Prestamos, Libros y Usuarios.

```sql
CREATE VIEW vista_prestamos_actuales AS
SELECT
    p.id AS id_prestamo,
    l.titulo AS nombre_libro,
    u.nombre AS nombre_usuario,
    p.fecha_prestamo
FROM
    Prestamos p
    INNER JOIN Libros l ON p.id_libro = l.id
    INNER JOIN Usuarios u ON p.id_usuario = u.id
WHERE
    p.fecha_devolucion IS NULL;
```

- En “create view” damos el nombre a la vista.
- Luego seleccionamos los campos:
  - p.id AS id_prestamo: Alias para el ID del préstamo en la tabla Prestamos.
  - l.titulo AS nombre_libro: Alias para el título del libro en la tabla Libros.
  - u.nombre AS nombre_usuario: Alias para el nombre del usuario en la tabla Usuarios.
  - p.fecha_prestamo: Fecha de préstamo desde la tabla Prestamos.

  De las tablas “prestamos” y “libros”.
- La condición en el “where” es que la fecha de devolución sea null, con esto nos aseguramos de que solo se incluyan los préstamos activos, es decir, aquellos que aún no tienen fecha de devolución registrada

Una vez que ya fue creada la vista, se puede consultar como si fuera una tabla normal.

SELECT * FROM vista_prestamos_actuales;

4. Disparadores (Triggers):

Los triggers son útiles para automatizar acciones en la base de datos en respuesta a ciertos eventos, ejecutándose de forma automática y transparente para el usuario, lo que ayuda a mantener la integridad y consistencia de los datos.
Para crear el trigger actualizar_fecha_devolucion, que se ejecutará antes de actualizar la tabla Prestamos para registrar la fecha actual como fecha_devolucion cuando un préstamo se devuelve, seguimos los siguientes pasos:

- Cambiamos el delimitador a // para poder incluir múltiples sentencias en el bloque del trigger. Después de definir el trigger, restauramos el delimitador a ;.
- Creamos el Trigger con el nombre correspondiente y luego:
  - BEFORE UPDATE ON Prestamos: Define que el trigger se activará antes de que se actualice alguna fila en la tabla Prestamos.
  - FOR EACH ROW: Especifica que el trigger se ejecutará para cada fila que se vea afectada por la operación de actualización.
  - BEGIN ... END: Contiene el cuerpo del trigger, donde implementamos la lógica.
  - IF NEW.fecha_devolucion IS NOT NULL THEN: Verifica si la nueva fecha_devolucion no es nula, lo que indica que se está actualizando para registrar la devolución del préstamo.
  - SET NEW.fecha_devolucion = NOW();: Asigna la fecha y hora actual (NOW()) a fecha_devolucion en la nueva fila (NEW), antes de que se realice la actualización en la tabla.

El script queda de la siguiente forma:

```
DELIMITER //
CREATE TRIGGER actualizar_fecha_devolucion
BEFORE UPDATE ON Prestamos
FOR EACH ROW
BEGIN
    IF NEW.fecha_devolucion IS NOT NULL THEN
        SET NEW.fecha_devolucion = NOW();
    END IF;
END //
DELIMITER ;
```

Una vez creado el trigger, cuando se actualice la columna fecha de devolución en la tabla Prestamos (por ejemplo, cuando se registra la devolución de un libro), automáticamente se registrará la fecha actual como fecha de devolución.

5. Transacciones:

Para implementar una transacción que registre un nuevo préstamo, actualice el estado del libro a "prestado" y registre la operación en una tabla de auditoría llamada AuditoriaPrestamos con los campos id_auditoria, id_prestamo, accion, fecha; los pasos a realizar son los siguientes:

- Transacción: Iniciar la transacción, esto permite que las operaciones subsecuentes sean tratadas como una unidad atómica.
- Se registra un nuevo préstamo.
- Se obtiene el ID del último prestamos registrado.
- Se actualiza el estado del libro a "prestado".
- Se registra la operación en la tabla AuditoriaPrestamos.
- Se confirma la transacción. En este paso se confirman todas las operaciones realizadas dentro de la transacción, asegurando que los cambios se hagan de manera permanente en la base de datos.

La transacción quedaría de esta forma:

```
START TRANSACTION;
INSERT INTO Prestamos (id_libro, id_usuario, fecha_prestamo)
VALUES (1, 2, NOW());
SET @id_prestamo = LAST_INSERT_ID();
UPDATE Libros
SET estado = 'prestado'
WHERE id = 1;
INSERT INTO AuditoriaPrestamos (id_prestamo, accion, fecha)
VALUES (@id_prestamo, 'Registro de préstamo', NOW());
COMMIT;
```

El nuevo ERD después de haber realizado las consultas es el siguiente: