Ejercicio 1: Dada una tabla clientes con id, nombre y apellidos, crear una api rest con la ruta base /clientes, con los siguientes métodos:

- getClientes: obtiene la lista de todos los clientes
- getCliente: dado un id, obtiene sus datos de clientes.
- insertaCliente: inserta los datos de un cliente.
- deleteCliente: dado un id de un cliente, lo elimina.
- actualizarCliente: dado un id de un cliente,actualiza sus datos con petición PUT.
- ActualizarClienteParcial: dado un id de un cliente,actualiza sus datos con petición PATCH.
- getClientesNombre: dado un nombre, obtiene la lista de todos los clientes cuyo nombre contenga el nombre a buscar.

Crear el controlador, servicio, repositorio y modelo de manera independiente en sus respectivos paquetes. Utilizar ResponseEntity.
Script:

```
CREATE TABLE `clientes` (
  `id` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `nombre` VARCHAR(20) NULL,
  `apellidos` VARCHAR(45) NULL
);
```

NOTA: El campo id es autonumérico.

Ejercicio 2: Dada la siguiente tabla product, crear una api rest con las operaciones CRUD: (Crear, leer todos, leer por id, actualizar y eliminar). Crear el controlador, servicio, repositorio y modelo de manera independiente en sus respectivos paquetes y con interfaces. Utilizar ResponseEntity. La ruta base es /product.

```
CREATE TABLE `product` (
  `id` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `name` VARCHAR(45) not NULL,
  `price` FLOAT NOT NULL
);
```

Añadir a continuación los siguientes métodos:

- Encontrar productos por nombre: Dado un nombre devuelve la lista de productos donde aparezca contenido dicho nombre.
- Encontrar productos por rango de precios: Dado dos precios (utilizad dos @pathVariable), obtener la lista de productos cuyo precio se encuentre en dicho rango.
- Insertar una lista de productos: Dada una lista de productos insertarlos en base de datos. Asegurándote de que los productos nuevos se inserten y los productos existentes se actualicen.

Ejercicio 3:

Desarrollar una API REST utilizando Spring Boot para gestionar los vehículos de una flota. La API permitirá realizar un CRUD completo (Crear, Leer, Actualizar y Eliminar) para gestionar la información de los vehículos. Además, deberá implementar métodos adicionales para realizar operaciones específicas relacionadas con el estado y el uso de los vehículos. La persistencia de datos se manejará mediante Hibernate con EntityManager y se almacenarán en una base de datos MariaDB. El controlador debe usar ResponseEntity como respuestas.

Script:

```
CREATE TABLE vehiculo (
   id BIGINT AUTO_INCREMENT PRIMARY KEY,
   marca VARCHAR(100) NOT NULL,
   modelo VARCHAR(100) NOT NULL,
   anyo INT NOT NULL,
   matricula VARCHAR(50) NOT NULL UNIQUE,
   estado VARCHAR(50) NOT NULL,
   kilometraje DOUBLE NOT NULL
);
```

Requerimientos de CRUD:

- Crear Vehículo: Permitir la creación de un nuevo vehículo con los atributos especificados. Debe devolver el vehículo insertado.
- Consultar Vehículo: Permitir la consulta de los detalles de un vehículo específico mediante su id
- Consultar la lista de vehículos: Devuelve la lista de todos los vehículos. Tener en cuenta si no hay ninguno.
- Actualizar Vehículo: Permitir la actualización de la información de un vehículo existente, dado su id. Actualización total.
- Eliminar Vehículo: Permitir la eliminación de un vehículo mediante su id. En caso de éxito, no mostrar nada.

Métodos Adicionales:

- Actualizar Estado de un vehículo: Endpoint para cambiar el estado de un vehículo (por ejemplo, cambiar de "Disponible" a "En Mantenimiento").
- Registrar Kilometraje: Endpoint para actualizar el kilometraje del vehículo.
- Consultar Vehículos por Estado: Endpoint para listar todos los vehículos que se encuentren en un estado específico (por ejemplo, todos los "En Uso").
- Filtrar Vehículos por Año: Endpoint para obtener una lista de vehículos fabricados dentro de un rango de dos años. Necesitará un año inicial y otro final para la consulta.

Ejercicio 4:

Desarrollar una API REST utilizando Spring Boot para gestionar hoteles. La API permitirá realizar un CRUD completo (Crear, Leer, Actualizar y Eliminar) para gestionar la información de los hoteles. Además, deberá implementar métodos adicionales para realizar operaciones específicas. La persistencia de datos se manejará mediante Hibernate con EntityManager y se almacenarán en una base de datos MariaDB. El controlador debe usar ResponseEntity como respuestas.

Script:

```
CREATE TABLE hotel (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL UNIQUE,
    direccion VARCHAR(255) NOT NULL,
    estrellas INT NOT NULL CHECK (estrellas BETWEEN 1 AND 5),
    telefono VARCHAR(50) NOT NULL,
    pagina_web VARCHAR(255)
);
```

Requisitos de la API:
Rutas Base: La ruta base para todas las operaciones será /hoteles.

Operaciones CRUD:

- POST /hoteles: Crear un nuevo hotel.
- GET /hoteles: Obtener una lista de todos los hoteles.
- GET /hoteles/{id}: Obtener un hotel específico por su ID.
- PUT /hoteles/{id}: Actualizar los detalles de un hotel existente.
- DELETE /hoteles/{id}: Eliminar un hotel por su ID.

Métodos Adicionales:

- GET /hoteles/estrellas/{min}/{max}: Obtener una lista de hoteles cuyo número de estrellas se encuentre dentro de un rango específico.
- GET /hoteles/telefono/{telefono}: Buscar un hotel por su número de teléfono. Devolver el primero usando streams.
- DELETE /hoteles/nombre/{nombre}: Eliminar todos los hoteles que contengan el nombre especificado.
- POST /hoteles/lista: insertar una lista de hoteles.

Ejercicio 5:

Desarrollar una API REST utilizando Spring Boot para gestionar los animales de un zoológico. La API permitirá realizar un CRUD completo (Crear, Leer, Actualizar y Eliminar) para gestionarlos. Además, deberá implementar métodos adicionales para realizar operaciones específicas. La persistencia de datos se manejará mediante Hibernate con EntityManager y se almacenarán en una base de datos MariaDB. El controlador debe usar ResponseEntity como respuestas.

Script:

```
CREATE TABLE animales (
id BIGINT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(255) NOT NULL UNIQUE,
especie VARCHAR(255) NOT NULL,
edad INT NOT NULL,
habitat VARCHAR(255) NOT NULL,
fechaIngreso DATE NOT NULL
);
```

Requisitos de la API:
Rutas Base: La ruta base para todas las operaciones será /api/animales.

Operaciones CRUD:

- POST : Crear un nuevo animal.
- GET : Obtener una lista de todos los animales.
- GET /{id}: Obtener un animal específico por su ID.
- PUT /{id}: Actualizar los detalles de un animal existente.
- DELETE /{id}: Eliminar un animal por su ID.

Métodos Adicionales:

- GET /especie/{especie}: Busca animales por especie
- GET /edad/{edad}: devuelve los animales que tienen 10 años. Solo interesa saber la especie, el nombre y su habitat.
- GET /recientes/{anyo}: devuelve los animales que han entrado en el zoo en los últimos x años. Solo interesa saber la fecha, la especie y el nombre.