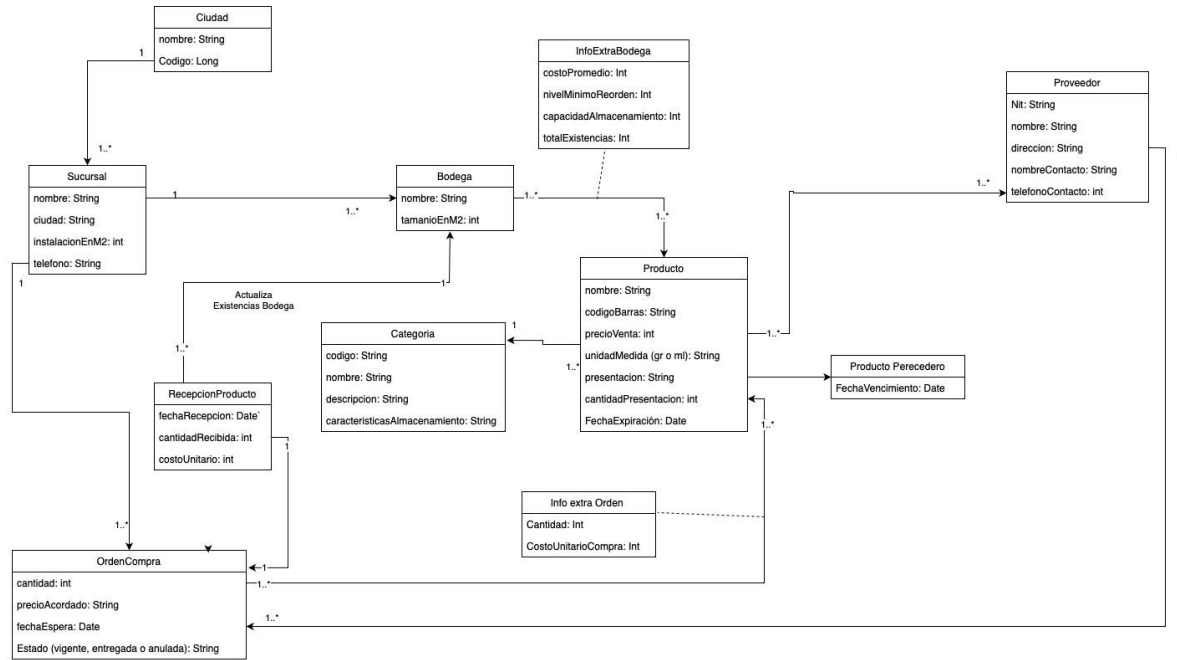


Entrega 1 – implementación

Modificaciones modelo UML



Cambios realizados:

1. Implementar clase hija a productos precederos
2. Implementar clase "Info Extra-bodega" para la cantidad mínima de productos de una bodega con respecto a el producto
3. Implementar clase ciudad
4. Implementar clase "Info extra-orden compra" para relacionar el costo unitario de la recepción de productos
5. Relación de producto-bodega: muchos a muchos
6. Relación de producto-Orden Compra: muchos a muchos

Modificaciones modelo Relacional

Ciudad	
Nombre	Código
NN	NN, ND, PK

Sucursal				
id_sucursal	nombre	Ciudad	instalacionM2	telefono
PK, NN, SA	NN, CK	FKCiudad.nombre, NN	NN	NN

Bodega			
id_bodega	Nombre	tamañoM2	id_sucursal
PK, SA, NN	NN, CK	NN	FKsucursal.id_sucursal, NN, ND

Producto								
id_producto	nombre	códigoBarras	precioVenta	presentacion	cantidadPresentacion	unidadMedida	expiracion	id_categoria
PK, SA, NN	NN, CK	NN, ND, CK	NN	NN	NN	NN	NN	FKcategoria.id_categoria, NN, ND

Categoria				
id_categoria	código	nombre	descripcion	caracteristicasAlmacenamiento
PK, SA, NN	NN, CK	NN	NN	NN

Proveedor					
id_proveedor	nif	nombre	direccion	nombreContacto	telefonoContacto
PK, SA, NN	NN, ND, CK	NN	NN	NN	NN

Orden Compra								
id_compra	cantidad	precioAcordado	fechaEspera	estado	id_proveedor	id_producto	id_sucursal	
PK, SA, NN, ND	NN	NN, ND	NN	NN	FKproveedor.id_proveedor, NN, ND	FKproducto.id_producto, NN, ND	FKsucursal.id_producto, NN, ND	

RecepcionProducto							
id_recepcion	fechaRecepcion	cantidadRecibida	costoUnitario	id_bodega	id_producto	id_bodega	
PK, SA, NN	NN	NN	NN	FKbodega.id_bodega, NN, ND	FKproducto.id_producto, NN, ND	FKbodega.id_bodega, NN, ND	

Orden_Producto	
id_producto	id_ordenCompra
FKproducto.id_producto, NN	FKordenCompra.id_compra, NN

Producto_proveedor	
id_producto	id_proveedor
FKproducto.id_producto, NN	FKproveedor.id_proveedor, NN

InfoExtraBodega					
CostoPromedio	NivelMinReOrden	capacidadAlmacenamiento	totalExistencias	id_bodega	id_producto
NN	NN	NN	NN	FKbodega.id_bodega, NN, ND	FKproducto.id_producto, NN, ND

ProductoPerecedero	
id_producto	fechaVencimiento
FKproducto.id_producto, NN	NN

InfoExtraOrden			
id_categoria	id_producto	cantidad	costoUnitario
PK, SA, NN	FKproducto.id_producto, NN	NN	NN

1. Identificación de entidades NUEVAS y MODIFICADAS:

- Entidad CIUDAD: En el modelo UML la clase ciudad representa las ciudades donde superAlpes tiene operaciones. Los atributos de la entidad son su nombre y su código. Para ambos se tiene de restricción NN (not null) el cual especifica que ningún elemento de esa columna debe estar vacío. Específicamente, el atributo Código que tiene ND (not duplicate) y es la llave identificadora PK (primary key)
- Entidad SUCURSAL: En el modelo UML, una sucursal es una entidad que representa una unidad de operación en diferentes ciudades. En el modelo relacional, los atributos de la entidad son nombre, ciudad, Instalación en m2 y teléfono, para todos se tiene de restricción NN (not null) el cual especifica que ningún elemento de esa columna debe estar vacío. Específicamente ID_SUCURSAL que es nuestra primary key (la llave identificadora) esta tiene como restricciones SA que significa que desempeña diferentes funciones para que el funcione y que todos los valores de la columna deben tener un valor. Además, el atributo ciudad es una foreign key (llave foránea) que hace referencia a la tabla CIUDAD donde llamamos a el nombre de la ciudad donde se encuentra esa sucursal.

- c. Entidad BODEGA: En el modelo UML, una bodega es una entidad que representa el lugar donde se almacenan los productos antes de ser enviados a su respectiva sucursal. Tiene atributos como nombre, tamaño en m2 y el id de la sucursal a la que está asociada, para todos se tiene como restricción NN (not null) el cual especifica que ningún elemento de esa columna debe estar vacío. Específicamente ID_BODEGA que es nuestra primary key (la llave identificadora) esta tiene como restricciones SA que significa que desempeña diferentes funciones para que el funcione y que todos los valores de la columna deben tener un valor.
- d. Entidad PRODUCTO En el modelo UML, un producto es una entidad que representa todos los artículos disponibles en el supermercado. Tiene atributos como nombre, precio de venta, presentación, cantidad de presentación, unidad de medida, peso, expiración y código de barras y el id de la categoría a la que está asociada, para todos se tiene como restricción NN (not null) el cual especifica que ningún elemento de esa columna debe estar vacío. Además en código de barras se tiene de restricción ND la cual nos indica que los valores dentro de esta columna no pueden ser repetidos ya que son únicos al producto que están representando. Específicamente ID_PRODUCTO que es nuestra primary key (la llave identificadora) esta tiene como restricciones SA que significa que desempeña diferentes funciones para que el funcione y que todos los valores de la columna tienen que tener un valor.
- e. Entidad INFOEXTRABODEGA: esta entidad relaciona la información que comparte una bodega y un producto. Sus atributos son: costo promedio de un producto, nivel mínimo de reorden, capacidad de almacenamiento, total de existencias, y se llama el id del producto al que está relacionado con un FK y a la bodega a la que está relacionada también con un FK, tienen la restricción NN.
- f. Entidad PRODUCTOPRECEDERO: esta entidad relaciona un producto precederos con su fecha de vencimiento. Como atributos esta su fecha de vencimiento y el id del producto que es una FK de producto. Ambos tienen la restricción NN.
- g. Entidad INFOEXTRAORDEN: esta entidad relaciona la información que comprarte una orden de compra y un producto. Sus atributos son: la cantidad y costo unitario ambos tienen la restricción NN. Y se utiliza el id del producto al que está relacionado la orden de compra con su id utilizando FK

2. Proceso de normalización para las Tablas nuevas

- a. CIUDAD: La tabla CIUDAD con los atributos Nombre y Código cumple con la BCNF ya que, todos los atributos contienen valores atómicos, satisfaciendo la 1NF. Además, el atributo Nombre depende completamente de la clave primaria Código, lo que asegura el cumplimiento de la 2NF. No existen dependencias transitivas entre los atributos, por lo que se cumple la 3NF.

Finalmente, la clave primaria Código es la única superllave, lo que garantiza que la tabla esté en BCNF.

- b. SUCURSAL: Para la tabla sucursal, todos los atributos contienen valores atómicos: nombre, ciudad, instalacionEnM2, teléfono, es decir que cumple con 1NF. Ahora bien, todos los atributos dependen completamente de la llave principal: id sucursal por lo que cumple con 2NF. Asimismo, no existen dependencias transitivas y por tanto cumple con 3NF. Por último, la llave primaria es la única super llave (cumple con BCNF).
- c. BODEGA: Para la tabla bodega, todos los atributos contienen valores atómicos: nombre, tamañoEnM2, lo que indica que cumple con 1NF. Asimismo, todos estos atributos dependen netamente de la llave principal: id bodega y cumple con 2NF. No existen dependencias transitivas (3NF), y la llave primaria es la única superllave (BCNF).
- d. PRODUCTO: Para la tabla producto, los atributos contienen valores atómicos: nombre, costo Bodega, precioVenta, presentación, cantidad Presentación, unidadMedida, volumen Empaque, peso Empaque, expiración, codigoBarras (1NF). De igual manera, todos los atributos dependen directamente de la llave principal: id_producto (2NF). También no se reflejan dependencias transitivas (3NF) y además la llave primaria es superllave (BCNF).
- e. INFOEXTRABODEGA: La tabla INFOEXTRABODEGA con los atributos CostoPromedio, NivelMinReOrden, capacidadAlmacenamiento, totalExistencias, id_bodega (FK) y id_producto (FK) cumple con la BCNF. Primero, todos los atributos contienen valores atómicos, lo que asegura que está en 1NF. Luego, los atributos dependen completamente de la combinación de las claves foráneas id_bodega y id_producto, cumpliendo con la 2NF. No existen dependencias transitivas, ya que ningún atributo no clave depende de otro atributo no clave, por lo que cumple con la 3NF. Finalmente, la clave primaria compuesta por id_bodega y id_producto es la única superllave, lo que garantiza que la tabla cumple con la BCNF.
- f. PRODUCTOPRECEDERO: La tabla PRODUCTO PRECEDERO con los atributos id_producto (FK) y FechaVencimiento cumple con la BCNF. En primer lugar, todos los atributos contienen valores atómicos, cumpliendo con la 1NF. El atributo FechaVencimiento depende completamente de la clave foránea id_producto, lo que garantiza el cumplimiento de la 2NF. No hay dependencias transitivas, ya que el único atributo no clave, FechaVencimiento, depende directamente de id_producto, cumpliendo con la 3NF. Finalmente, id_producto es la única superllave, lo que asegura que la tabla también cumple con la BCNF.
- g. INFOEXTRAORDEN: La tabla INFO EXTRA ORDEN con los atributos id_categoria (PK), id_producto (FK), cantidad y costoUnitario cumple con la BCNF. Todos los atributos contienen valores atómicos, satisfaciendo la 1NF. Los atributos cantidad y costoUnitario dependen completamente de la clave primaria compuesta por id_categoria y id_producto, cumpliendo con la 2NF.

No existen dependencias transitivas, garantizando la 3NF, y la clave primaria es la única superllave, por lo que también cumple con la BCNF.

Documentación creación de tablas en SQL DEVELOPER

Creación de tablas:

1. CIUDAD: Define una tabla con una columna nombre que permite hasta 2 caracteres y una columna código como clave primaria.

```
CREATE TABLE CIUDAD(  
  nombre VARCHAR2(2) NOT NULL,  
  codigo NUMBER(25) PRIMARY KEY NOT NULL );
```

2. SUCURSAL: Define una tabla de sucursales con un identificador único id_sucursal como clave primaria.

```
CREATE TABLE SUCURSAL (  
  id_sucursal NUMBER(25) PRIMARY KEY NOT NULL,  
  nombre VARCHAR2(20) NOT NULL,  
  intalacionM2 NUMBER DEFAULT 25 NOT NULL,  
  telefono NUMBER DEFAULT 25 NOT NULL);
```

```
ALTER TABLE SUCURSAL  
ADD codigo_ciudad NUMBER(25) NOT NULL;
```

```
ALTER TABLE SUCURSAL  
ADD CONSTRAINT fk_ciudad_sucursal  
FOREIGN KEY (codigo_ciudad)  
REFERENCES CIUDAD(codigo);
```

3. BODEGA: Define una tabla para las bodegas con id_bodega como clave primaria.

```
ALTER TABLE BODEGA
ADD id_sucursal NUMBER(25) NOT NULL;

ALTER TABLE BODEGA
ADD CONSTRAINT fk_sucursal_bodega
FOREIGN KEY (id_sucursal)
REFERENCES SUCURSAL(id_sucursal);
```

4. PROVEEDORES: Define una tabla para los proveedores con id_proveedor como clave primaria.

```
CREATE TABLE PROVEEDORES(
id_proveedor  NUMBER(25) PRIMARY KEY NOT NULL,
nit NUMBER(25) NOT NULL,
nombre VARCHAR2(25) NOT NULL,
direccion VARCHAR2(25) NOT NULL,
nombre_contacto VARCHAR(25) NOT NULL,
telefono_contacto VARCHAR (25) NOT NULL);
```

5. CATEGORIA: Define una tabla de categorías de productos con id_categoria como clave primaria.

```
CREATE TABLE CATEGORIA(
id_categoria  NUMBER(25) PRIMARY KEY NOT NULL,
codigo VARCHAR(25) NOT NULL,
nombre VARCHAR(25) NOT NULL,
descripcion VARCHAR(25),
caracteristicas_almacenamiento VARCHAR2(25) NOT NULL);
```

6. PRODUCTO: Define una tabla para productos con id_producto como clave primaria.

```
CREATE TABLE PRODUCTO (  
  id_producto NUMBER(25) PRIMARY KEY NOT NULL,  
  nombre VARCHAR(255) NOT NULL,  
  codigoBarras VARCHAR(255) NOT NULL,  
  precioVenta NUMBER DEFAULT 25 NOT NULL,  
  presentacion VARCHAR(255) NOT NULL,  
  cantidadPresentacion NUMBER DEFAULT 25 NOT NULL,  
  unidadMedida VARCHAR(255) NOT NULL,  
  expiracion DATE NOT NULL);  
ALTER TABLE PRODUCTO  
ADD id_categoria NUMBER(25) NOT NULL;  
  
ALTER TABLE PRODUCTO  
ADD CONSTRAINT fk_categoria_producto  
FOREIGN KEY (id_categoria)  
REFERENCES CATEGORIA(id_categoria);
```

7. ORDENCOMPRA: Define una tabla para órdenes de compra con id_compra como clave primaria.

```

ALTER TABLE ORDENCOMPRA
ADD id_sucursal NUMBER(25) NOT NULL;

ALTER TABLE ORDENCOMPRA
ADD id_producto NUMBER(25) NOT NULL;

ALTER TABLE ORDENCOMPRA
ADD id_proveedor NUMBER(25) NOT NULL;

ALTER TABLE ORDENCOMPRA
ADD CONSTRAINT fk_sucursal_ordencompra
FOREIGN KEY (id_sucursal)
REFERENCES SUCURSAL(id_sucursal);

ALTER TABLE ORDENCOMPRA
ADD CONSTRAINT fk_producto_ordencompra
FOREIGN KEY (id_producto)
REFERENCES PRODUCTO(id_producto);

ALTER TABLE ORDENCOMPRA
ADD CONSTRAINT fk_proveedor_ordencompra
FOREIGN KEY (id_proveedor)
REFERENCES PROVEEDORES(id_proveedor);

```

```

CREATE TABLE ORDENPRODUCTO (
id_producto NUMBER(25) NOT NULL,
id_compra NUMBER(25) NOT NULL,
CONSTRAINT fk_producto_ordenproducto FOREIGN KEY (id_producto) REFERENCES PRODUCTO(id_producto),
CONSTRAINT fk_compra_ordenproducto FOREIGN KEY (id_compra) REFERENCES ORDENCOMPRA(id_compra),
CONSTRAINT pk_ordenproducto PRIMARY KEY (id_producto, id_compra));

```

8. RECEPCIONPRODUCTO: Define una tabla para la recepción de productos con id_recepcion como clave primaria.


```
CREATE TABLE RECEPCIONPRODUCTO (  
  id_recepcion NUMBER(25) PRIMARY KEY NOT NULL,  
  fechaRecepcion DATE NOT NULL,  
  cantidadRecibida INT NOT NULL,  
  costoUnitario INT NOT NULL);
```

```
ALTER TABLE RECEPCIONPRODUCTO  
ADD id_bodega NUMBER(25) NOT NULL;  
  
ALTER TABLE RECEPCIONPRODUCTO  
ADD id_producto NUMBER(25) NOT NULL;  
  
ALTER TABLE RECEPCIONPRODUCTO  
ADD CONSTRAINT fk_bodega_recepcion  
FOREIGN KEY (id_bodega)  
REFERENCES BODEGA(id_bodega);  
  
ALTER TABLE RECEPCIONPRODUCTO  
ADD CONSTRAINT fk_producto_recepcion  
FOREIGN KEY (id_producto)  
REFERENCES PRODUCTO(id_producto);
```

9. INFOEXTRABODEGA: Define una tabla para información adicional sobre las bodegas.

```
CREATE TABLE INFOEXTRABODEGA (  
  costopromedio INT NOT NULL,  
  nivelMinReorden INT NOT NULL,  
  capacidadAlmacenamiento INT NOT NULL,  
  totalExistencias INT NOT NULL);  
  
ALTER TABLE INFOEXTRABODEGA  
  ADD id_bodega NUMBER(25) NOT NULL;  
  
ALTER TABLE INFOEXTRABODEGA  
  ADD id_producto NUMBER(25) NOT NULL;  
  
ALTER TABLE INFOEXTRABODEGA  
  ADD CONSTRAINT fk_bodega_infoextrabodega  
  FOREIGN KEY (id_bodega)  
  REFERENCES BODEGA(id_bodega);  
  
ALTER TABLE INFOEXTRABODEGA  
  ADD CONSTRAINT fk_producto_infoextrabodega  
  FOREIGN KEY (id_producto)  
  REFERENCES PRODUCTO(id_producto);  
  
ALTER TABLE INFOEXTRABODEGA  
  ADD CONSTRAINT pk_infoextrabodega  
  PRIMARY KEY (id_bodega, id_producto);
```

10. PRODUCTOPERECEDERO: Define una tabla para productos perecederos.

```
CREATE TABLE PRODUCTOPERECEDERO (  
  fechaVencimiento DATE NOT NULL);
```

```
ALTER TABLE PRODUCTOPERECEDERO
ADD id_producto NUMBER(25) NOT NULL;

ALTER TABLE PRODUCTOPERECEDERO
ADD CONSTRAINT pk_fk_productoperecedero
PRIMARY KEY (id_producto);

ALTER TABLE PRODUCTOPERECEDERO
ADD CONSTRAINT fk_productoperecedero_producto
FOREIGN KEY (id_producto)
REFERENCES PRODUCTO(id_producto);
```

11. INFOEXTRAORDEN: Define una tabla para información adicional sobre órdenes.

```
CREATE TABLE INFOEXTRAORDEN(
cantidad INT NOT NULL,
costoUnitario INT NOT NULL);
```

```

ALTER TABLE INFOEXTRAORDEN
ADD id_categoria NUMBER(25) NOT NULL;

ALTER TABLE INFOEXTRAORDEN
ADD id_producto NUMBER(25) NOT NULL;

ALTER TABLE INFOEXTRAORDEN
ADD CONSTRAINT fk_categoria_infoextraorden
FOREIGN KEY (id_categoria)
REFERENCES CATEGORIA(id_categoria);

ALTER TABLE INFOEXTRAORDEN
ADD CONSTRAINT fk_producto_infoextraorden
FOREIGN KEY (id_producto)
REFERENCES PRODUCTO(id_producto);

ALTER TABLE INFOEXTRAORDEN
ADD CONSTRAINT pk_infoextraorden
PRIMARY KEY (id_categoria, id_producto);

```

12. ORDENPRODUCTO: Define una tabla de relación entre productos y órdenes de compra, con una clave primaria compuesta por id_producto y id_compra.

```

CREATE TABLE ORDENPRODUCTO (
id_producto NUMBER(25) NOT NULL,
id_compra NUMBER(25) NOT NULL,
CONSTRAINT fk_producto_ordenproducto FOREIGN KEY (id_producto) REFERENCES PRODUCTO(id_producto),
CONSTRAINT fk_compra_ordenproducto FOREIGN KEY (id_compra) REFERENCES ORDENCOMPRA(id_compra),
CONSTRAINT pk_ordenproducto PRIMARY KEY (id_producto, id_compra));

```

13. PRODUCTOPROVEEDOR: Define una tabla de relación entre productos y proveedores, con una clave primaria compuesta por id_producto y id_proveedor.

```

CREATE TABLE PRODUCTOPROVEEDOR(
id_producto NUMBER(25),
id_proveedor NUMBER(25),
CONSTRAINT fk_producto_productoproveedor FOREIGN KEY (id_producto) REFERENCES PRODUCTO(id_producto),
CONSTRAINT fk_proveedor_productoproveedor FOREIGN KEY (id_proveedor) REFERENCES PROVEEDORES(id_proveedor),
CONSTRAINT pk_productoproveedor PRIMARY KEY (id_producto, id_proveedor));

```

Modificación de Tablas(ALTER)

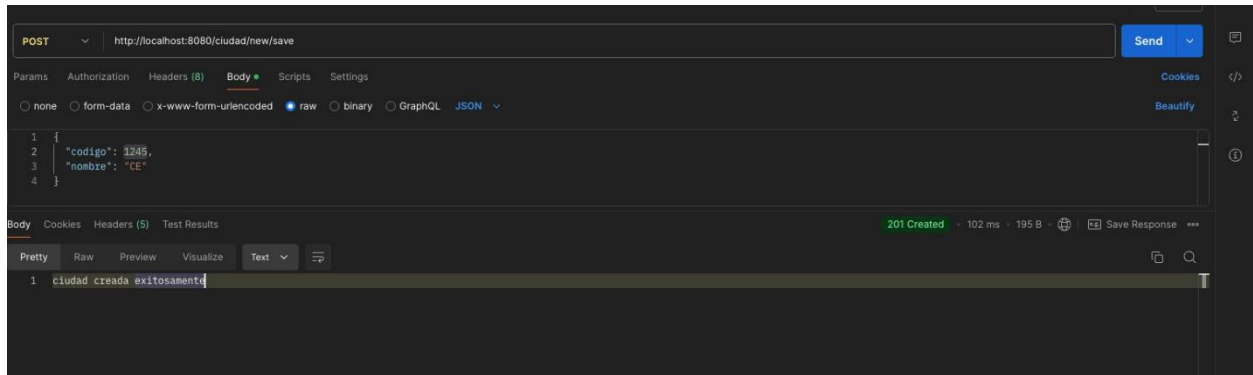
Se agregan columnas y claves foráneas para establecer relaciones entre las tablas, como la relación entre sucursales y ciudades, o entre productos y categorías.

Eliminación de Tablas

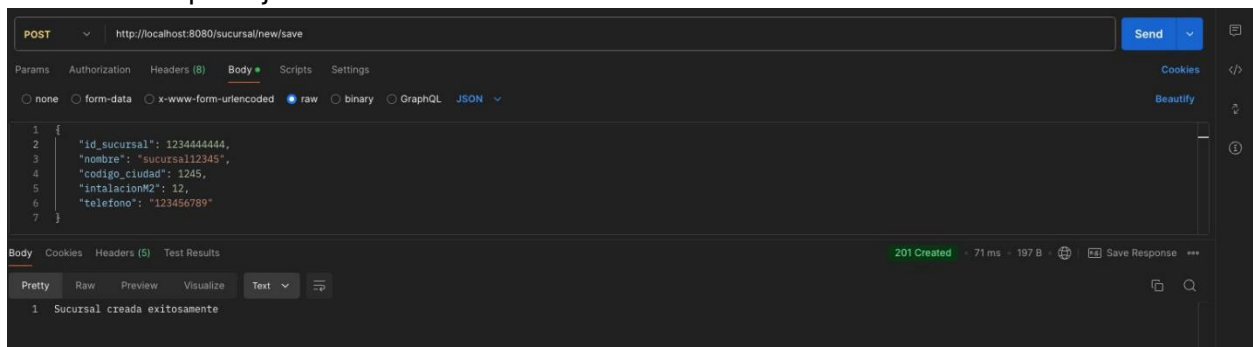
Finalmente, las tablas se eliminan en el orden inverso al que fueron creadas para asegurar que todas las dependencias sean correctamente eliminadas. La cláusula CASCADE CONSTRAINTS asegura que todas las restricciones de integridad referencial se eliminen junto con las tablas.

Documentación requerimientos funcionales:

1. RF1: Este código implementa una funcionalidad básica para manejar la entidad Ciudad, que se mapea a la tabla CIUDAD en la base de datos. La entidad Ciudad tiene dos atributos principales: código, que es la clave primaria y nombre, que representa el nombre de la ciudad. El repositorio CiudadRepository extiende JpaRepository, proporcionando métodos estándar para operaciones CRUD, además de un método personalizado insertarCiudad, que inserta una nueva ciudad en la base de datos utilizando una consulta SQL nativa. El controlador CiudadController expone dos consultas: uno para obtener todas las ciudades (GET /ciudad) y otro para crear una nueva ciudad (POST /ciudad/new/save). En el primer caso, el método devuelve una lista de ciudades o un error si la operación falla; en el segundo, guarda una nueva ciudad en la base de datos y responde con un mensaje de éxito o un error en caso de fallar. Los métodos están anotados para gestionar transacciones y asegurar la persistencia de los datos en la base de datos.



- RF2: Este código define una estructura completa para manejar la entidad Sucursal, la cual se mapea a la tabla SUCURSAL en la base de datos. La clase Sucursal incluye varios atributos, como id_sucursal, nombre, codigo_ciudad, intalacionM2, y telefono, que representan las propiedades de una sucursal. El repositorio SucursalRepository, que extiende JpaRepository, proporciona métodos para realizar operaciones CRUD, además de un método personalizado insertarSucursal para insertar nuevas sucursales utilizando una consulta SQL nativa. El controlador SucursalController expone dos endpoints: uno para obtener todas las sucursales (GET /sucursal) y otro para crear una nueva sucursal (POST /sucursal/new/save). El método sucursales() devuelve una colección de todas las sucursales o un error si ocurre un problema durante la recuperación de datos. El método createSucursal() recibe los datos de una sucursal a través de un cuerpo de solicitud JSON y llama al método insertarSucursal del repositorio para guardarla en la base de datos. En caso de éxito, devuelve un estado 201 CREATED; si ocurre un error, devuelve un mensaje descriptivo junto con un estado 500 INTERNAL SERVER ERROR.

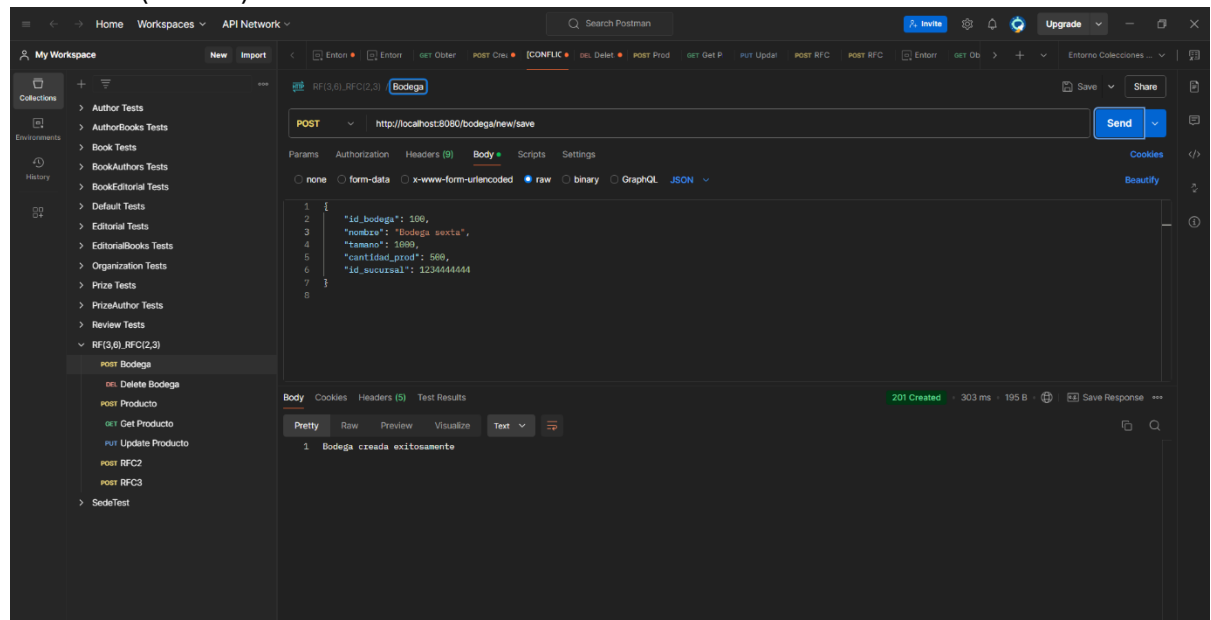


- RF3: Este código implementa la funcionalidad básica para manejar la entidad Bodega, que se mapea a la tabla BODEGA en la base de datos. La entidad Bodega tiene varios atributos principales: id_bodega, nombre, tamaño, cantidad_prod, y id_sucursal, que indica la sucursal a la cual está asociada la bodega. Es importante recordar que una bodega debe estar vinculada a una sucursal, y una sucursal puede tener una o más bodegas asociadas.

El controlador **BodegaController** expone dos consultas principales:

Crear Bodega: Crear una nueva bodega (POST /bodega/new/save): Este método recibe los datos de la bodega en el cuerpo de la solicitud y guarda la nueva bodega en la base de datos, asociándola a una sucursal existente. Si la creación es exitosa, se devuelve un mensaje de confirmación; en caso contrario, se envía un mensaje de error.

Postman (Create):

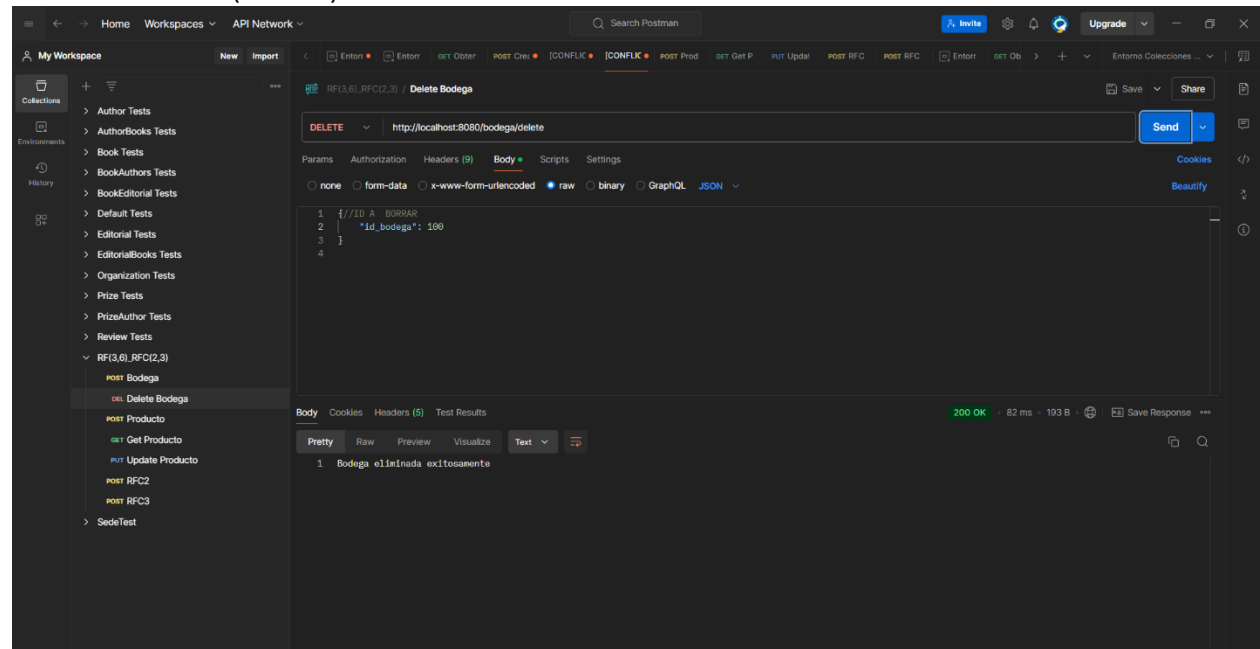


SQL Developer(Create):

	ID_BODEGA	NOMBRE	TAMANO	CANTIDAD_PROD	ID_SUCURSAL
1	2001	Bodega Norte	2000	150	1001
2	101	Bodega Carlos	1000	500	1234444444
3	3007	Bodega Sucursal Sur	3500	800	567890123
4	100	Bodega sexta	1000	500	1234444444
5	1	Bodega Central	5000	2000	1001
6	2	Bodega Norte	3000	1200	1001
7	3	Bodega Sur	4000	1500	1001

Borrar Bodega(Delete): Borrar una bodega (DELETE /bodega/delete/{id}): Este método permite eliminar una bodega existente de la base de datos, identificada por su `id_bodega`. Si la operación de borrado es exitosa, se devuelve un mensaje de éxito; de lo contrario, se envía un error si la bodega no es encontrada.

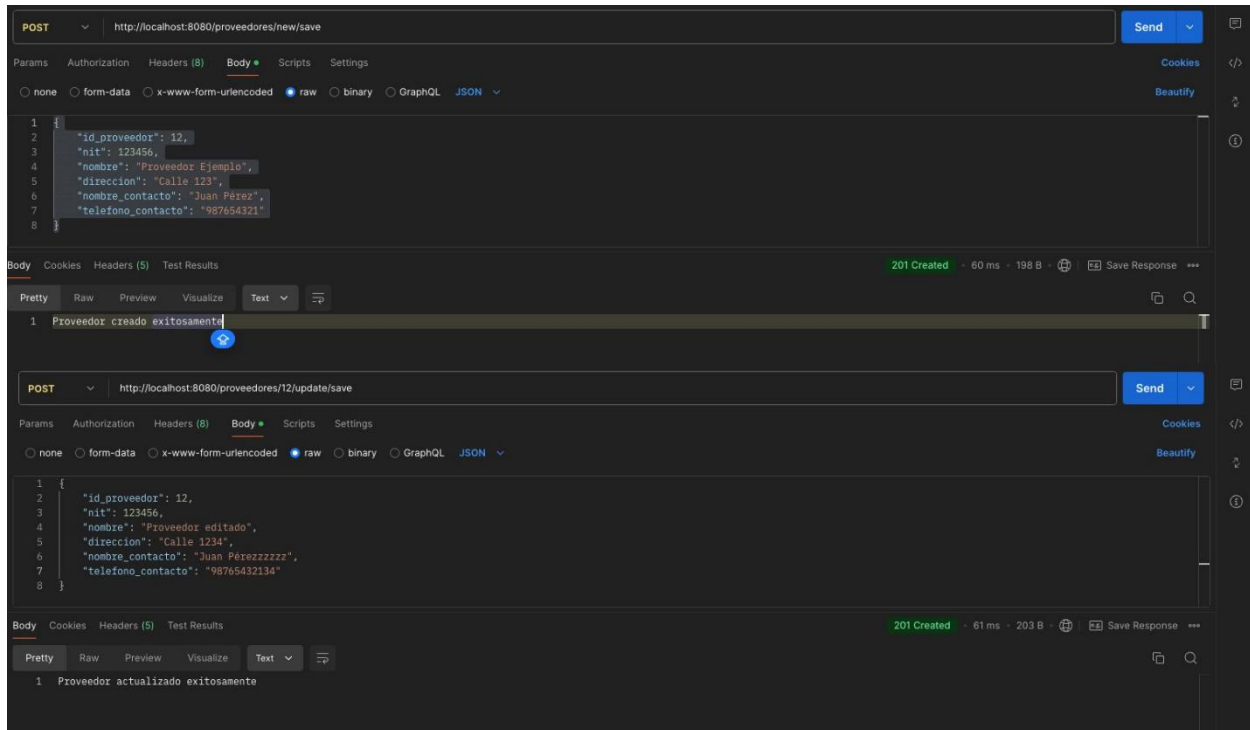
Postman(Delete):



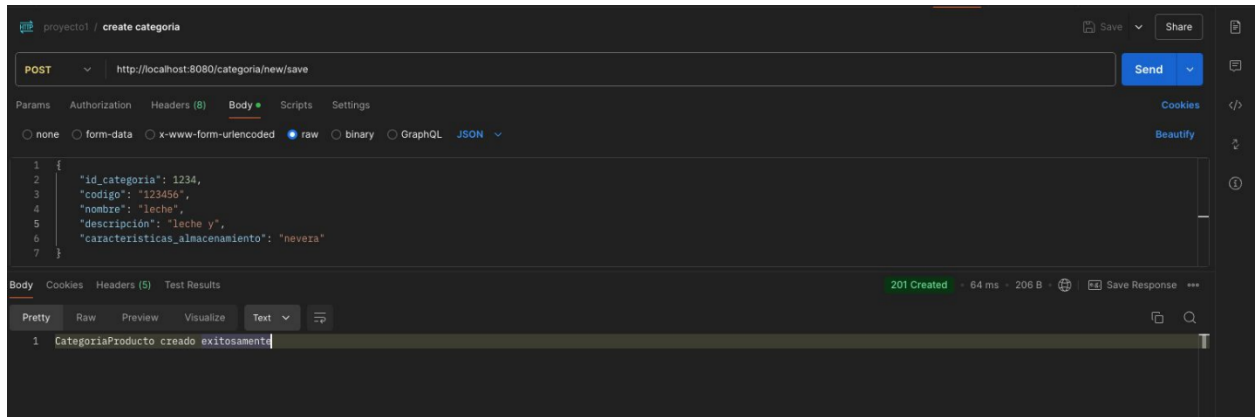
sql(borrado):

ID_BODEGA	NOMBRE	TAMANO	CANTIDAD_PROD	ID_SUCURSAL
1	2001 Bodega Norte	2000	150	1001
2	101 Bodega Carlos	1000	500	1234444444
3	3007 Bodega Sucursal Sur	3500	800	567890123
4	1 Bodega Central	5000	2000	1001
5	2 Bodega Norte	3000	1200	1001
6	3 Bodega Sur	4000	1500	1001

- RF4: Este código define la estructura y funcionalidad para manejar la entidad Proveedores, que se mapea a la tabla PROVEEDORES en la base de datos. La clase Proveedores incluye varios atributos, como `id_proveedor`, `nit`, `nombre`, `direccion`, `nombre_contacto`, y `telefono_contacto`, que representan las propiedades de un proveedor. La clase también incluye un constructor parametrizado para inicializar estas propiedades, así como los métodos `get` y `set` correspondientes para acceder y modificar los atributos. El repositorio `ProveedoresRepository` extiende `JpaRepository` y proporciona métodos personalizados para insertar y actualizar registros en la tabla PROVEEDORES mediante consultas SQL nativas. El método `insertarProveedor` permite agregar un nuevo proveedor a la base de datos, mientras que el método `actualizarProveedor` permite modificar un registro existente basándose en el `id_proveedor`. En caso de éxito, devuelve un estado `201 CREATED`; si ocurre un error, devuelve un mensaje descriptivo junto con un estado `500 INTERNAL SERVER ERROR`.



5. RF5: Este código define la estructura y funcionalidad para manejar la entidad `CategoriaProducto`, la cual se mapea a la tabla `CATEGORIA` en la base de datos. La clase `CategoriaProducto` incluye varios atributos como `id_categoria`, `codigo`, `nombre`, `descripcion`, y `caracteristicas_almacenamiento`, que representan las propiedades de una categoría de producto. El repositorio `CategoriaProductoRepository` proporciona métodos personalizados para insertar y leer categorías en la tabla `CATEGORIA` utilizando consultas SQL nativas. El método `insertarCategoria` permite agregar una nueva categoría de producto, mientras que el método `leerCategoria` permite recuperar una categoría específica utilizando su `id_categoria`. El controlador `CategoriaProductoController` expone dos endpoints: uno para obtener todas las categorías (`GET /categoria`) y otro para crear una nueva categoría (`POST /categoria/new/save`). El método `categoriaProducto()` devuelve una lista de todas las categorías o un error si ocurre un problema durante la recuperación de datos. El método `createCategoriaProducto()` recibe los datos de una categoría a través de un cuerpo de solicitud JSON y llama al método `insertarCategoria` del repositorio para guardarla en la base de datos. En caso de éxito, devuelve un estado `201 CREATED`; si ocurre un error, devuelve un mensaje de error junto con un estado `500 INTERNAL SERVER ERROR`.



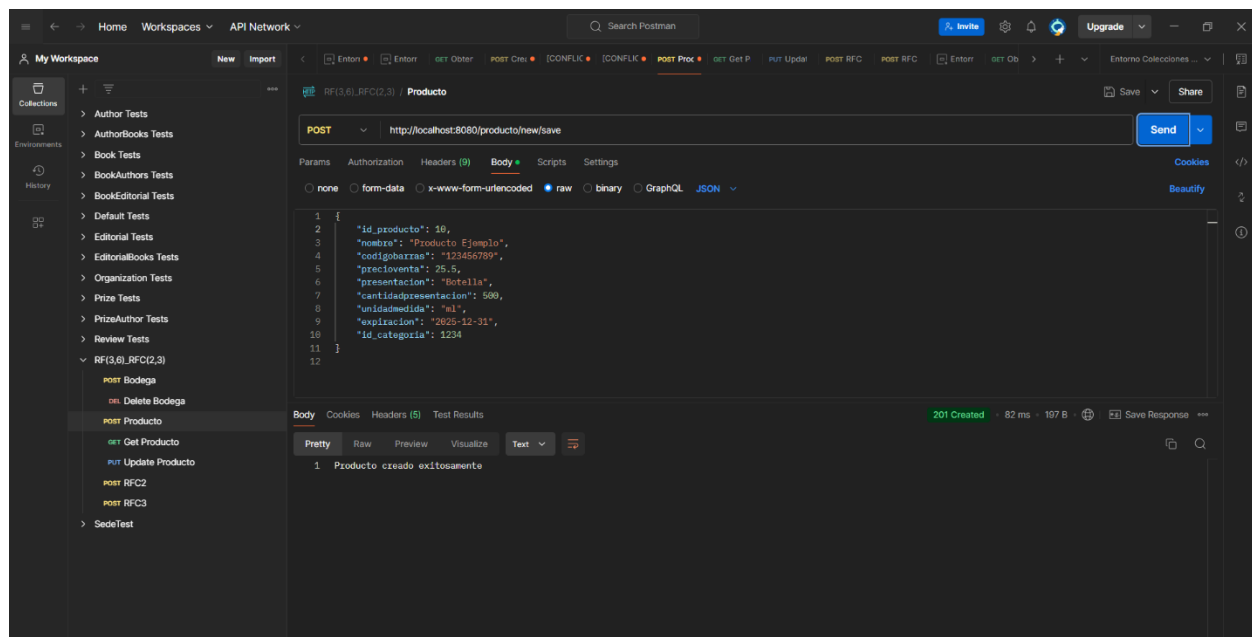
6. RF6:

Este código implementa una funcionalidad básica para manejar la entidad **Producto**, que se mapea a la tabla **PRODUCTO** en la base de datos. La entidad **Producto** tiene varios atributos principales: `id_producto`, `nombre`, `codigobarras`, `precioventa`, `presentacion`, `cantidadpresentacion`, `unidadmedida`, `expiracion`, y `id_categoria`, que representa la categoría a la que pertenece el producto.

El controlador **ProductoController** expone tres consultas principales

Crear Producto: (POST /producto/new/save): Este método recibe los datos del producto en el cuerpo de la solicitud y guarda el producto en la base de datos, asociándolo a una categoría existente. Si la creación es exitosa, responde con un mensaje de éxito; en caso contrario, devuelve un error.

Postman (Create):

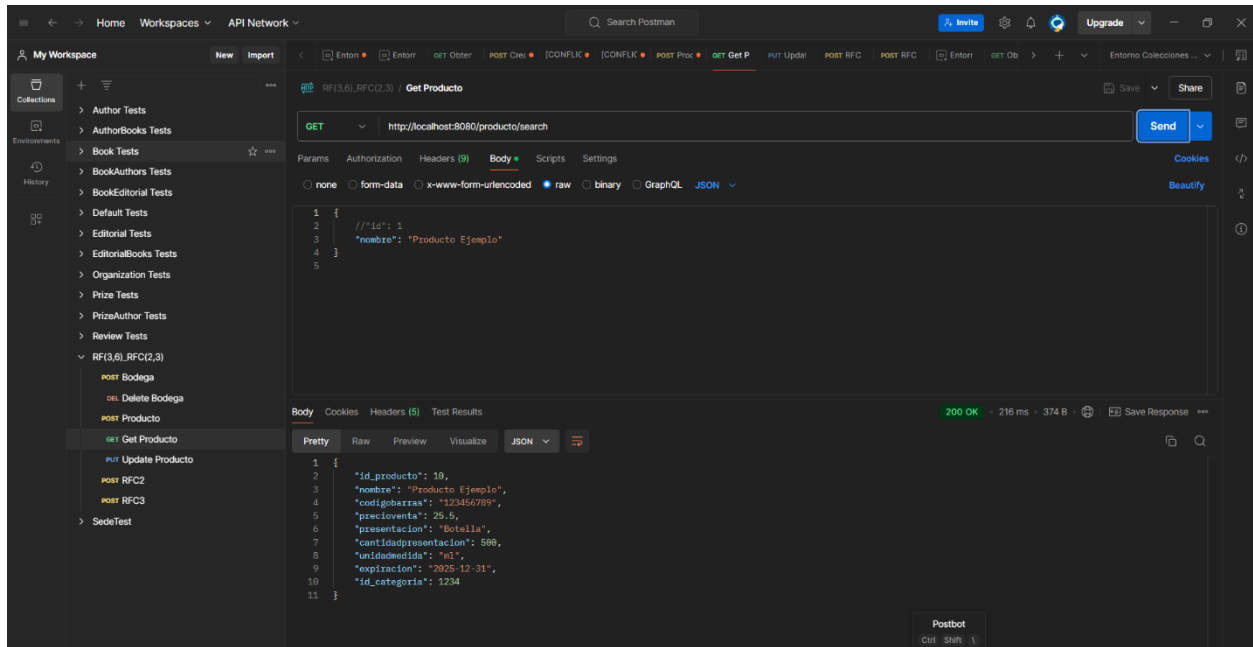


SQL:

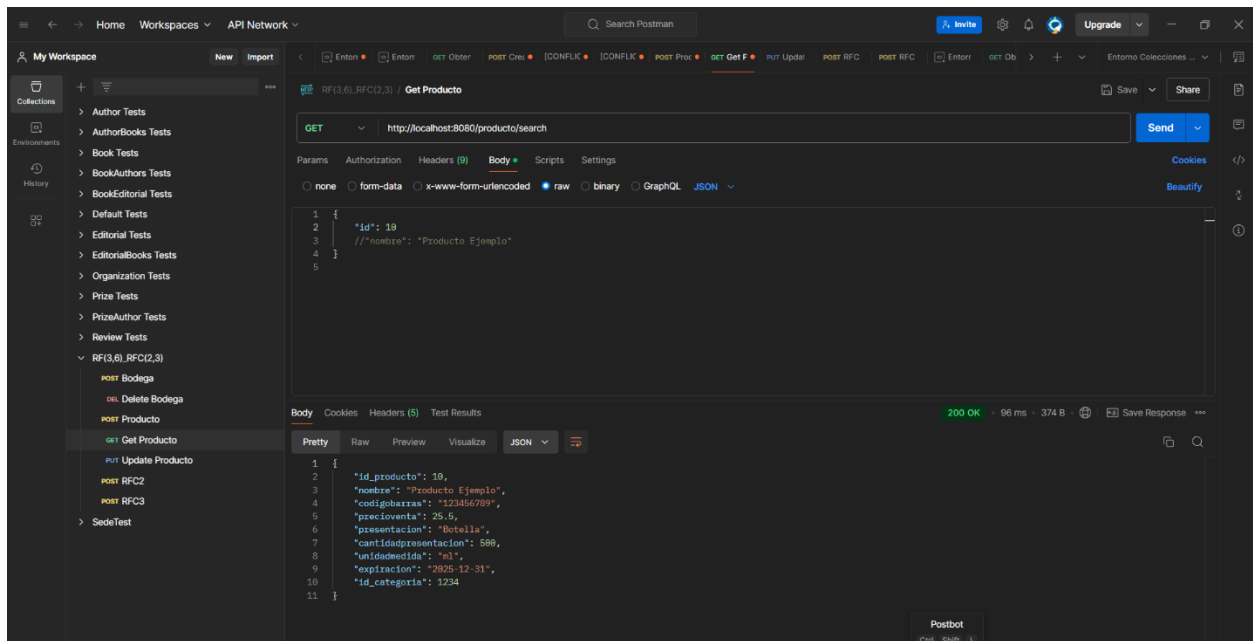
	ID_PRODUCTO	NOMBRE	CODIGOBARRAS	PRECIOVENTA	PRESENTACION	CANTIDADPRESENTACION	UNIDADMEDIDA
1	3001	Producto A	987654321		100 Caja Grande		15 kg
2	4001	Producto A	1234567890		100 Caja Grande		15 kg
3	4002	Producto B	1234567891		50 Caja Mediana		10 kg
4	4003	Producto C	1234567892		75 Caja Pequeña		5 kg
5	3002	Producto B	123456789		50.5 Caja Mediana		10 kg
6	3003	Producto C	192837465		75 Caja Pequeña		5 kg
7	10	Producto Ejemplo	123456789		25.5 Botella		500 ml

Get Producto: (GET /producto/search): Permite obtener los detalles de un producto proporcionando su id o nombre. El método busca el producto en la base de datos y devuelve la información correspondiente o un error si no se encuentra.

Postman con nombre:

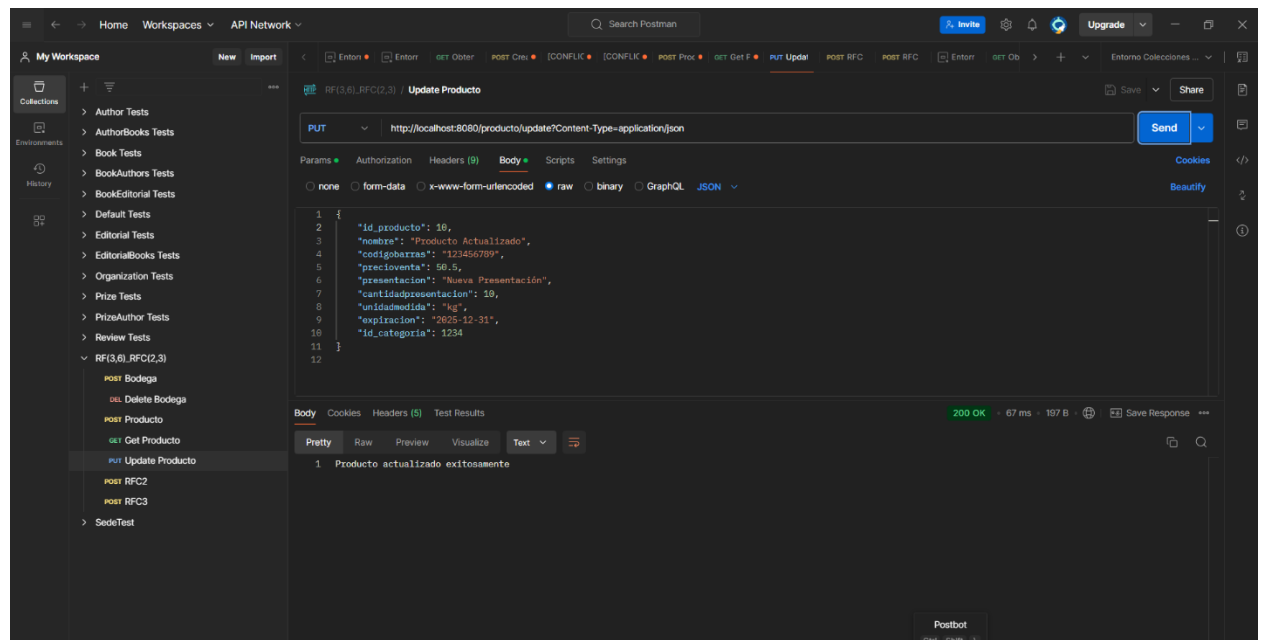


Postman con id:



UPDATE PRODUCTO: (PUT /producto/update): Este método permite la actualización de los datos de un producto existente en la base de datos, identificado por su `codigobarras`. Los campos actualizables incluyen `nombre`, `precioventa`, `presentacion`, `cantidadpresentacion`, `unidadmedida`, `expiracion`, y la `id_categoria`. Si la actualización es exitosa, se devuelve un mensaje de confirmación, de lo contrario, se envía un mensaje de error.

Postman:



SQL:

7	10 Producto Actualizado 123456789	50.5 Nueva Presentación	10 kg
---	-----------------------------------	-------------------------	-------

7. RF7:

Este requerimiento se centra en la funcionalidad para la creación de órdenes de compra vinculadas a sucursales. Al igual que otras transacciones importantes, las órdenes de compra se dividen en dos componentes: **el encabezado** y **el detalle**.

Encabezado de la Orden de Compra:

El encabezado contiene información general sobre la orden de compra, que incluye:

Fecha de creación: La fecha en que se genera la orden de compra. Esta fecha se establece automáticamente al momento de crear la orden, y no es modificable por el usuario.

Sucursal: El usuario selecciona la sucursal desde donde se va a realizar la compra de una lista de sucursales disponibles. Esto permite identificar a qué sucursal se debe destinar el pedido.

Proveedor: El proveedor de los productos es escogido por el usuario de una lista preexistente de proveedores. Esto define quién será el encargado de proveer los productos a la sucursal.

Fecha esperada de entrega: Esta es la fecha en la cual se espera que los productos solicitados sean entregados a la sucursal. El usuario ingresa esta fecha manualmente al crear la orden de compra.

Detalle de la Orden de Compra:

El detalle se refiere a la lista de productos que se incluyen en la orden. Cada producto tiene las siguientes propiedades:

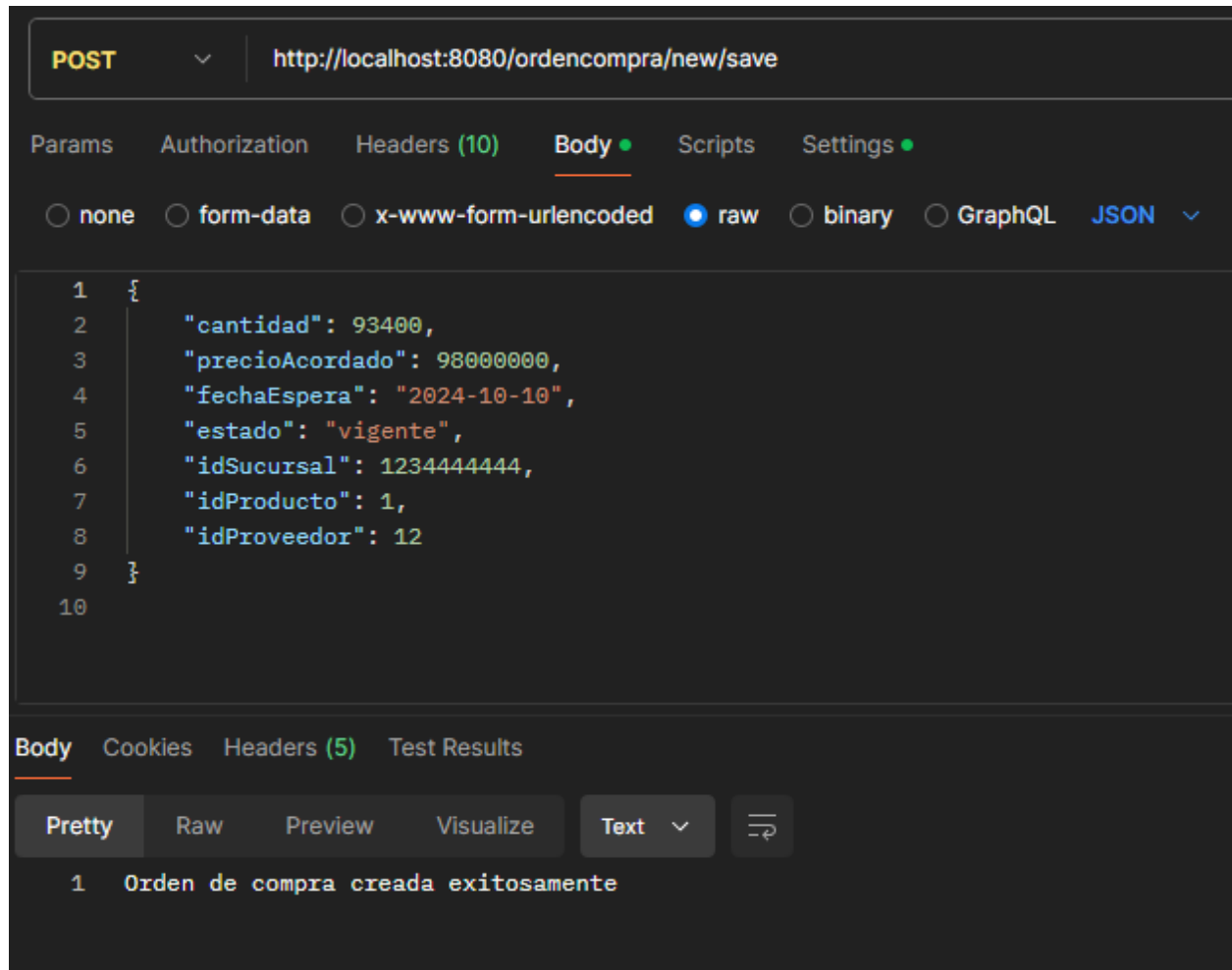
Productos: Se selecciona uno o más productos de los que ya existen en el catálogo de productos.

Cantidad: El usuario ingresa la cantidad específica de cada producto que se va a solicitar.

Precio acordado: El precio de cada producto, el cual puede ser ingresado directamente por el usuario. Este campo permite reflejar acuerdos de precios que podrían diferir de los precios estándar.

Estado de la Orden de Compra:

Una vez que la orden de compra es creada, su estado inicial se define como "**vigente**". Este estado indica que la orden está activa y a la espera de ser procesada y entregada. Posteriormente, la orden puede cambiar de estado conforme avanza su ciclo de vida (por ejemplo, a "entregada" o "anulada").



8. RF8:

Este requerimiento se enfoca en el proceso de actualización del estado de una **orden de compra** a "anulada" dentro del sistema de gestión de órdenes. La funcionalidad permite que, en ciertos casos, las órdenes de compra puedan ser canceladas, pero con restricciones específicas para mantener la integridad del proceso de adquisición de productos.

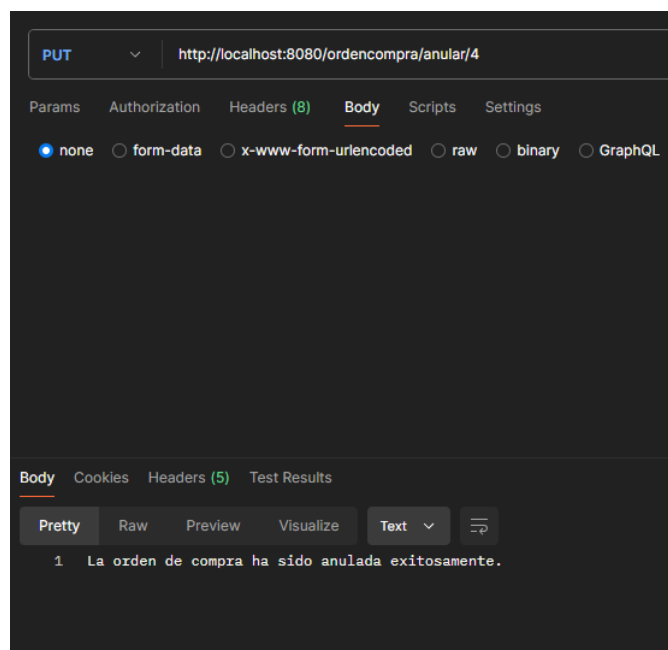
Proceso de Anulación de una Orden de Compra:

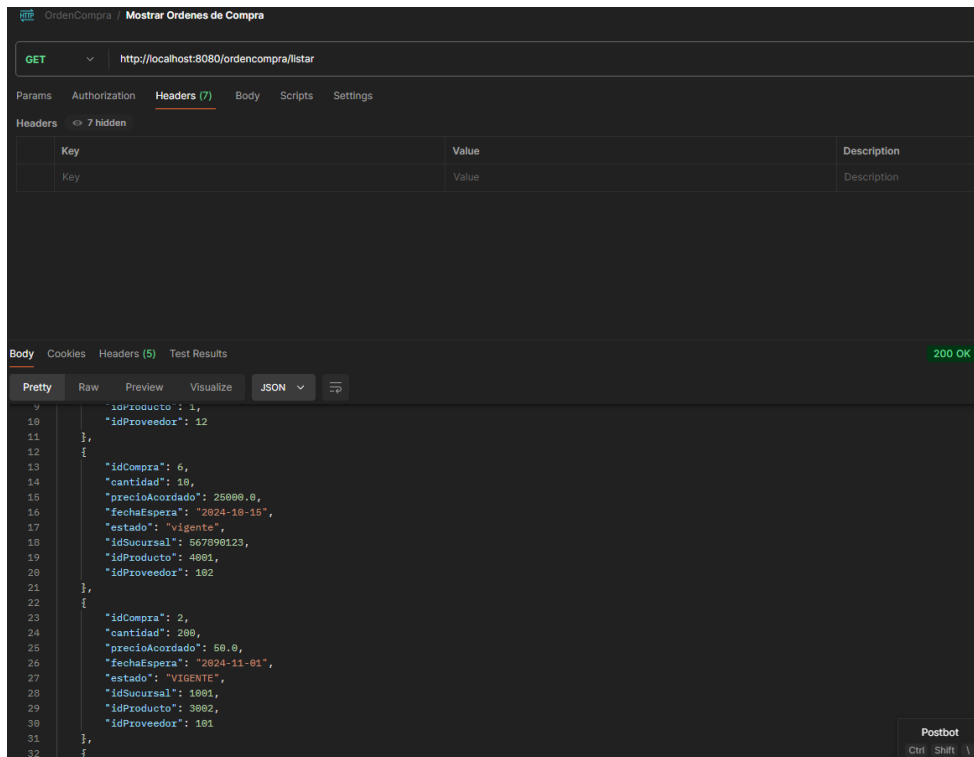
Ingreso del número de la orden de compra: El usuario ingresa o selecciona el número de la orden de compra que desea anular. Este número es clave, ya que permite identificar de forma única la orden que será modificada.

Verificación del estado de la orden: Antes de realizar cualquier cambio en el estado de la orden de compra, el sistema verifica que la orden se encuentre en estado "**vigente**". Solo las órdenes en estado "vigente" pueden ser anuladas.

Restricción para órdenes entregadas: Si la orden de compra está en estado "**entregada**", no puede ser anulada. Este control asegura que no se puedan modificar o cancelar órdenes que ya han sido cumplidas por el proveedor.

Actualización del estado a "anulada": Si la orden de compra cumple con las condiciones mencionadas (estado vigente y no entregada), el sistema actualiza el estado de la orden a "**anulada**". Este cambio de estado indica que la orden de compra ya no es válida y no será procesada ni entregada.





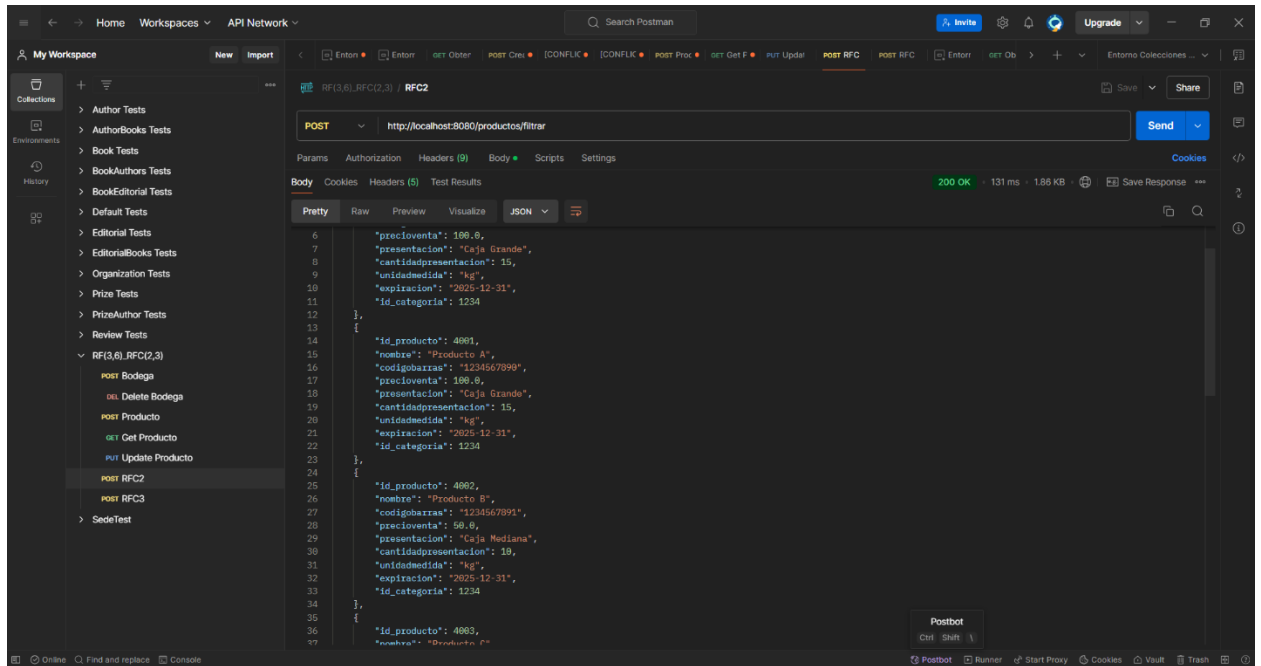
Documentaciones funcionales de consulta:

1. RFC 1:

Este código implementa la funcionalidad para mostrar el índice de ocupación de cada una de las bodegas en una sucursal, que se mapea a la tabla BODEGA en la base de datos. La bodega tiene atributos clave como `id_bodega`, nombre, tamaño (capacidad de almacenamiento), y `cantidad_prod` (cantidad de productos almacenados).

El controlador **BodegaController** expone una consulta principal:

- Mostrar el índice de ocupación de las bodegas** (POST `/bodegas/ocupacion`): Este método recibe una lista de productos en el cuerpo de la solicitud, para los cuales se quiere conocer el porcentaje de ocupación. A partir de esta lista, el sistema calcula el volumen ocupado por esos productos en las bodegas correspondientes, comparándolo con la capacidad total de cada bodega. El resultado es un informe que presenta el porcentaje de ocupación de cada bodega.

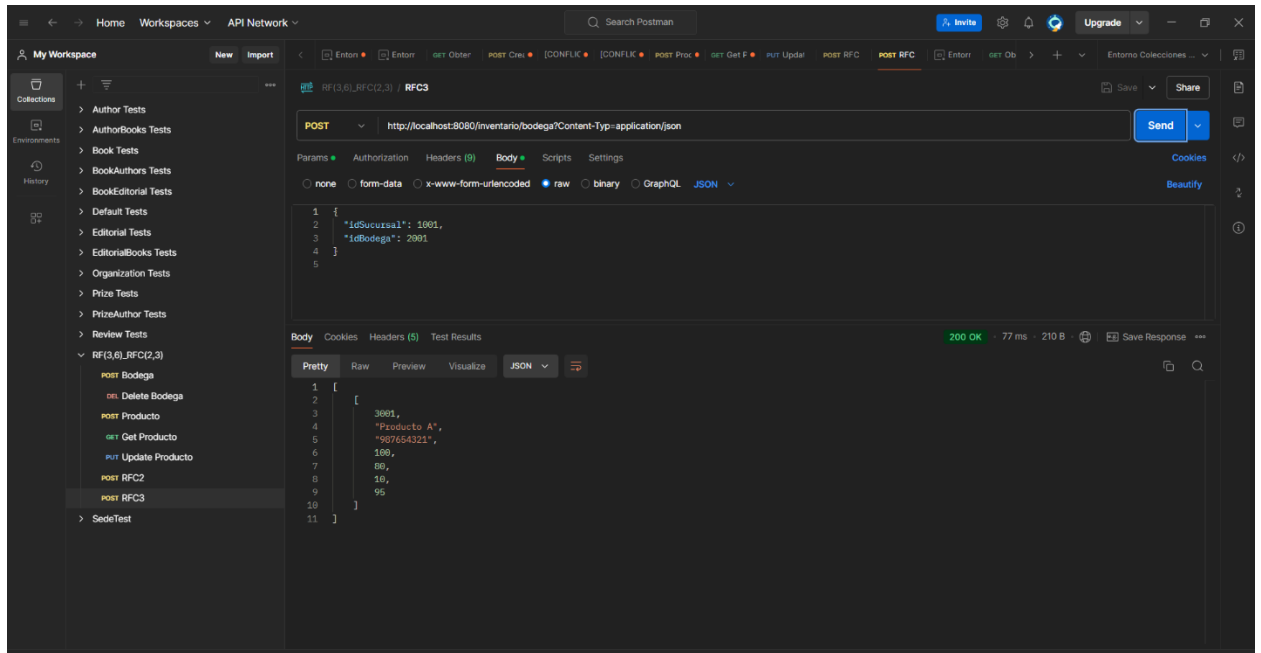


3. RFC 3:

Este código implementa la funcionalidad para obtener un reporte de inventario de productos en una bodega específica asociada a una sucursal, la cual se mapea a las tablas **PRODUCTO**, **INFOEXTRABODEGA**, **BODEGA** y **SUCURSAL** en la base de datos. La entidad **Producto** contiene información básica de los productos, mientras que **InfoExtraBodega** almacena información adicional relacionada con el inventario en la bodega, como la cantidad actual de productos, la cantidad mínima requerida y el costo promedio.

Obtener el inventario de productos en una bodega (POST

/inventario/bodega): Este método recibe un JSON en el cuerpo de la solicitud, que contiene el ID de la sucursal (`idSucursal`) y el ID de la bodega (`idBodega`). A partir de estos parámetros, el sistema realiza una consulta en la base de datos para listar los productos disponibles en la bodega correspondiente, junto con la información solicitada de cantidad, cantidad mínima y costo promedio.



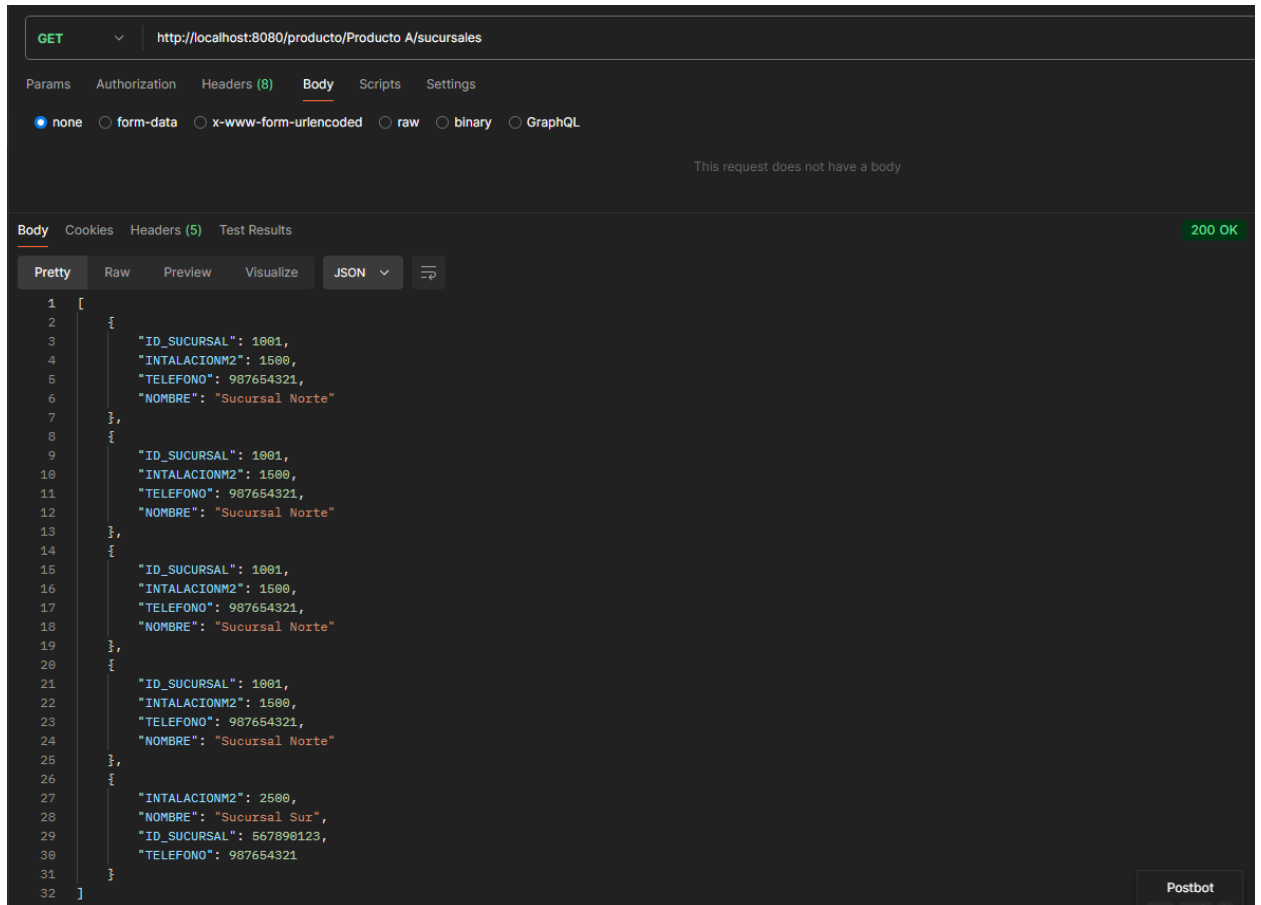
4. RFC 4:

Este requerimiento tiene como objetivo permitir a los usuarios consultar todas las **sucursales** de **SuperAndes** donde un producto específico está disponible. La búsqueda se puede realizar tanto por el **identificador** del producto como por su **nombre**.

Consulta por producto: El usuario ingresa el identificador o nombre del producto.

Consulta en bodegas: El sistema busca en las bodegas asociadas a las sucursales y verifica si el producto está disponible, es decir, si existe en cantidades mayores a cero.

Resultados: Se genera una lista de las sucursales que tienen el producto en stock en alguna de sus bodegas, incluyendo detalles como el nombre de la sucursal y posiblemente el nivel de inventario del producto en cada bodega.



5. RFC 5:

Este requerimiento permite listar todos los **productos** cuyo stock en una bodega está por debajo del **nivel mínimo de reorden**, lo que indica que es necesario generar una nueva orden de compra para reabastecerlos.

Identificación de productos con stock bajo: El sistema busca productos en las bodegas cuyo nivel de inventario esté por debajo del nivel mínimo de reorden.

Información detallada: Por cada producto, se debe mostrar:

- **Nombre e identificador** del producto.
- La **bodega** donde el stock es bajo.
- Las **sucursales** asociadas a las bodegas.
- Los **proveedores** a los que se les ha comprado el producto en el pasado.
- Las **cantidades actuales** del producto en esas bodegas.

GET http://localhost:8080/productos/reorden

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
-----	-------	-------------

Body Cookies Headers (5) Test Results 200 OK

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "ID_BODEGA": 3007,
4     "ID_PRODUCTO": 4001,
5     "NOMBRE_PROVEEDOR": null,
6     "NOMBRE_BODEGA": "Bodega Sucursal Sur",
7     "CANTIDAD_ACTUAL": 500,
8     "ID_SUCURSAL": 567890123,
9     "NOMBRE_PRODUCTO": "Producto A",
10    "NOMBRE_SUCURSAL": "Sucursal Sur",
11    "ID_PROVEEDOR": null
12  },
13  {
14    "CANTIDAD_ACTUAL": 300,
15    "NOMBRE_BODEGA": "Bodega Central",
16    "NOMBRE_PROVEEDOR": null,
17    "ID_SUCURSAL": 1001,
18    "NOMBRE_SUCURSAL": "Sucursal Norte",
19    "ID_PRODUCTO": 3002,
20    "NOMBRE_PRODUCTO": "Producto Actualizado",
21    "ID_BODEGA": 1,
22    "ID_PROVEEDOR": null
23  },
24  {
25    "NOMBRE_PROVEEDOR": null,
26    "CANTIDAD_ACTUAL": 350,
27    "ID_SUCURSAL": 1001,
28    "NOMBRE_SUCURSAL": "Sucursal Norte",
29    "ID_PRODUCTO": 3002,
30    "NOMBRE_PRODUCTO": "Producto Actualizado",
31    "ID_BODEGA": 2,
32    "NOMBRE_BODEGA": "Bodega Norte",
33    "ID_PROVEEDOR": null
34  },
35  ]
```

Postbot
Ctrl Shift \

Entrega Proyecto 2

RF10 - REGISTRAR INGRESO DE PRODUCTOS A LA BODEGA:

```

// 1. Eliminar Recepción de Producto
@Modifying
@Transactional
@Query(value = "DELETE FROM RECEPCIONPRODUCTO WHERE ID_RECEPCION = :idRecepcion", nativeQuery = true)
void eliminarRecepcionProducto(@Param("idRecepcion") Integer idRecepcion);

/*// 2. Eliminar Producto de la Bodega
@Modifying
@Transactional
@Query(value = "DELETE FROM BODEGA_PRODUCTO WHERE ID_BODEGA = :idBodega AND ID_PRODUCTO = :idProducto", nativeQuery = true)
void eliminarProductoDeBodega(@Param("idBodega") Integer idBodega, @Param("idProducto") Integer idProducto); */

// 3. Actualizar el estado de la Orden de Compra a 'anulada'
@Modifying
@Transactional
@Query(value = "UPDATE ORDENCOMPRA SET ESTADO = 'anulada' WHERE ID_COMPRA = :idCompra", nativeQuery = true)
void actualizarEstadoOrdenCompra(@Param("idCompra") Integer idCompra);

// 4. Insertar Recepción de Producto
@Modifying
@Transactional
@Query(value = "INSERT INTO RECEPCIONPRODUCTO (ID_RECEPCION, FECHARECEPCION, CANTIDADRECIBIDA, COSTOUNITARIO, ID_BODEGA, ID_PRODUCTO) " +
    "VALUES (:idRecepcion, SYSDATE, :cantidadRecibida, :costoUnitario, :idBodega, :idProducto)", nativeQuery = true)
void insertarRecepcionProducto(@Param("idRecepcion") Integer idRecepcion, @Param("cantidadRecibida") Integer cantidadRecibida,
    @Param("costoUnitario") Double costoUnitario, @Param("idBodega") Integer idBodega, @Param("idProducto") Integer idProducto);

// 5. Actualizar la cantidad en Bodega
@Modifying
@Transactional
@Query(value = "UPDATE BODEGA_PRODUCTO SET CANTIDAD = CANTIDAD + :cantidadRecibida WHERE ID_BODEGA = :idBodega AND ID_PRODUCTO = :idProducto", nativeQuery = true)
void actualizarCantidadBodega(@Param("idBodega") Integer idBodega, @Param("idProducto") Integer idProducto, @Param("cantidadRecibida") Integer cantidadRecibida);

// 6. Actualizar el estado de la Orden de Compra a 'ENTREGADA'
@Modifying
@Transactional
@Query(value = "UPDATE ORDENCOMPRA SET ESTADO = 'ENTREGADA' WHERE ID_COMPRA = :idOrden", nativeQuery = true)
void actualizarEstadoOrdenEntregada(@Param("idOrden") Integer idOrden);

@PostMapping("/bodega/recepcion")
@Transactional(rollbackFor = Exception.class)
public ResponseEntity<?> getInsertarProductoBodega(@RequestParam("v_id_orden") Integer v_id_orden, @RequestParam("v_id_producto") Integer v_id_producto,
    @RequestParam("v_id_bodega") Integer v_id_bodega, @RequestParam("v_cantidad_recibida") Integer v_cantidad_recibida,
    @RequestParam("v_costo_unitario") Double v_costo_unitario, @RequestParam("v_cantidad_existente") Integer v_cantidad_existente,
    @RequestParam("v_cantidad_orden") Integer v_cantidad_orden) {

    try {
        // 1. Eliminar Recepción de Producto
        bodegaRepository.eliminarRecepcionProducto(idRecepcion:2);

        // 2. Eliminar Producto de la Bodega
        //bodegaRepository.eliminarProductoDeBodega(v_id_bodega, v_id_producto);

        // 3. Actualizar el Estado de la Orden de Compra a 'anulada'
        bodegaRepository.actualizarEstadoOrdenCompra(v_id_orden);

        // 4. Insertar Recepción de Producto
        bodegaRepository.insertarRecepcionProducto(idRecepcion:2, v_cantidad_recibida, v_costo_unitario, v_id_bodega, v_id_producto);

        // 5. Actualizar la Cantidad en Bodega
        bodegaRepository.actualizarCantidadBodega(v_id_bodega, v_id_producto, v_cantidad_recibida);

        // 6. Actualizar el Estado de la Orden de Compra a 'ENTREGADA'
        bodegaRepository.actualizarEstadoOrdenEntregada(v_id_orden);

        return ResponseEntity.status(HttpStatus.OK).body("transaccion correcta");
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error en la transaccion " + e.getMessage());
    }
}
}
}

```

El código define varios métodos que permiten realizar operaciones transaccionales en una base de datos mediante consultas SQL nativas. Estas operaciones incluyen la eliminación de registros de recepción de productos, la inserción de nuevos registros en la tabla de recepciones, y la actualización del estado de las órdenes de compra, cambiándolo "anulada" o "ENTREGADA" según sea necesario.

Además, se proporciona un método para actualizar la cantidad de productos en la bodega, sumando las cantidades recibidas. Todo esto se implementa de manera que las transacciones se realicen de forma segura y eficiente, con el uso de anotaciones `@Transactional` para garantizar que las operaciones se ejecuten correctamente y se reviertan en caso de error.

El controlador principal, que gestiona la recepción de productos en la bodega, coordina todas estas operaciones en una transacción única. Si ocurre alguna excepción, se activa un rollback automático y se envía un mensaje de error al cliente, asegurando la integridad de los datos y un manejo adecuado de posibles fallos en la transacción.

RFC6:

```
@Service
public class RecepcionProductoService {

    @Autowired
    private RecepcionProductoRepository recepcionProductoRepository;

    @Transactional (isolation = Isolation.SERIALIZABLE)
    public List<Map<String, Object>> obtenerDocumentosIngreso(Integer idSucursal, Integer idBodega) {

        try {
            // Temporizador de 30 segundos para simular la espera de concurrencia
            Thread.sleep(millis:30000);

            // Llamada al método del repositorio
            return recepcionProductoRepository.buscarDocumentosIngreso(idSucursal, idBodega);
        } catch (InterruptedException e) {
            System.out.println("El temporizador fue interrumpido: " + e.getMessage());
            throw new RuntimeException(message:"El temporizador fue interrumpido.", e);
        }
    }
}
```

Este requerimiento permite consultar los documentos de ingreso de productos a una bodega específica, con un enfoque en garantizar la consistencia de la información en un entorno concurrente. Específicamente, hace lo siguiente:

Consulta Documentos de Ingreso: Obtiene un listado de los documentos de recepción de productos en una bodega específica de una sucursal, mostrando la siguiente información:

- El nombre de la sucursal y de la bodega.
- Un listado de los documentos de ingreso realizados en los últimos 30 días, incluyendo el número del documento, la fecha de recepción y el nombre del proveedor involucrado.

Transacción Serializable: Se implementa con el mayor nivel de aislamiento (Serializable) para evitar problemas de concurrencia, como lecturas sucias, lecturas no repetibles, o fantasmas. Esto asegura que la consulta se realice sobre una vista consistente de los datos.

Rollback en Caso de Error: Si no se puede realizar la consulta, la transacción se revierte y se informa al usuario.

Temporizador de 30 Segundos: La transacción incluye un temporizador de 30 segundos para simular una demora y evaluar cómo se comporta el sistema en condiciones de concurrencia.

Este requerimiento asegura la consistencia de los datos y permite identificar de manera precisa los documentos de recepción que se han registrado recientemente, proporcionando una visión detallada sobre los movimientos de inventario en las bodegas.

RFC7:

```
@Service
public class RecepcionProductoService {

    @Autowired
    private RecepcionProductoRepository recepcionProductoRepository;

    @Transactional (isolation = Isolation.READ_COMMITTED)
    public List<Map<String, Object>> obtenerDocumentosIngreso(Integer idSucursal, Integer idBodega) {

        try {
            // Temporizador de 30 segundos para simular la espera de concurrencia
            Thread.sleep(millis:30000);

            // Llamada al método del repositorio
            return recepcionProductoRepository.buscarDocumentosIngreso(idSucursal, idBodega);
        } catch (InterruptedException e) {
            System.out.println("El temporizador fue interrumpido: " + e.getMessage());
            throw new RuntimeException(message:"El temporizador fue interrumpido.", e);
        }
    }
}
```

Este requerimiento permite consultar los documentos de ingreso de productos a una bodega específica, con un enfoque en garantizar la consistencia de la información en un entorno concurrente. Específicamente, hace lo siguiente:

Consulta Documentos de Ingreso: Obtiene un listado de los documentos de recepción de productos en una bodega específica de una sucursal, mostrando la siguiente información:

- El nombre de la sucursal y de la bodega.
- Un listado de los documentos de ingreso realizados en los últimos 30 días, incluyendo el número del documento, la fecha de recepción y el nombre del proveedor involucrado.

Transacción Read Committed: Se implementa con un nivel de aislamiento Read Committed, que asegura que solo se lean datos que han sido confirmados, evitando lecturas sucias. Sin embargo, este nivel permite que puedan ocurrir lecturas no repetibles

y fenómenos de fantasmas, ya que otras transacciones pueden modificar o insertar datos después de que se ha realizado la lectura inicial, pero antes de que la transacción actual se complete. Esto proporciona un equilibrio entre consistencia y rendimiento en entornos con alta concurrencia.

Rollback en Caso de Error: Si no se puede realizar la consulta, la transacción se revierte y se informa al usuario.

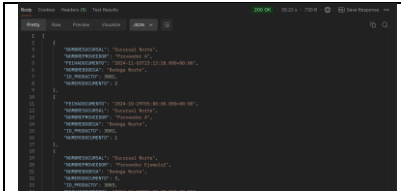

Temporizador de 30 Segundos: La transacción incluye un temporizador de 30 segundos para simular una demora y evaluar cómo se comporta el sistema en condiciones de concurrencia.

Este requerimiento asegura la consistencia de los datos y permite identificar de manera precisa los documentos de recepción que se han registrado recientemente, proporcionando una visión detallada sobre los movimientos de inventario en las bodegas.

Caso de Concurrencia 1:

1. Línea de tiempo

RFC6 - serializable	TIEMPO S	RF10
Ejecutar consulta RFC6 <pre>service public class RecpcionProductoService { @Transactional private RecpcionProductoRepository recpcionProductoRepository; @Transactional (isolation = Isolation.SERIALIZABLE) public List<RecepcionProducto> obtenerRecepcionProducto(Integer idBodega, Integer idRecepcion) { try { // Temporizador de 30 segundos para simular la espera de concurrencia Thread.sleep(30000); // Llamar al método de recepción return recpcionProductoRepository.obtenerRecepcionProducto(idBodega, idRecepcion); } catch (InterruptedException e) { System.out.println("El temporizador fue interrumpido " + e.getMessage()); throw new RuntimeException("Mensaje: El temporizador fue interrumpido.", e); } } }</pre>	T1	
	T2	Ejecutar query rf10 <pre>// 1. Eliminar Recepción de Producto @Transactional @Query(value = "DELETE FROM RECEPCION_PRODUCTO WHERE ID_RECEPCION = :idRecepcion", nativeQuery = true) void eliminarRecepcionProducto(@Param("idRecepcion") Integer idRecepcion); // 2. Eliminar Producto de la Bodega @Transactional @Query(value = "DELETE FROM BODEGA_PRODUCTO WHERE ID_BODEGA = :idBodega AND ID_PRODUCTO = :idProducto", nativeQuery = true) void eliminarProductoBodega(@Param("idBodega") Integer idBodega, @Param("idProducto") Integer idProducto); // 3. Actualizar el estado de la Orden de Compra a 'avilada' @Transactional @Query(value = "UPDATE ORDENCOMPRAS SET ESTADO = 'avilada' WHERE ID_ORDENCOMPRAS = :idOrdencompra", nativeQuery = true) void actualizarEstadoOrdenCompra(@Param("idOrdencompra") Integer idOrdencompra); // 4. Insertar Recepción de Producto @Transactional @Query(value = "INSERT INTO RECEPCION_PRODUCTO (ID_RECEPCION, FECHA_RECEPCION, CANTIDAD_RECEPCION, CANTIDAD_ORDEN, ID_BODEGA, ID_PRODUCTO) " + "VALUES (:idRecepcion, SYSDATE(), :cantidadRecepcion, :cantidadOrden, :idBodega, :idProducto)", nativeQuery = true) void insertarRecepcionProducto(@Param("idRecepcion") Integer idRecepcion, @Param("cantidadRecepcion") Integer cantidadRecepcion, @Param("cantidadOrden") Integer cantidadOrden, @Param("idBodega") Integer idBodega, @Param("idProducto") Integer idProducto); // 5. Actualizar la cantidad en Bodega @Transactional @Query(value = "UPDATE BODEGA_PRODUCTO SET CANTIDAD = CANTIDAD - :cantidadRecepcion WHERE ID_BODEGA = :idBodega AND ID_PRODUCTO = :idProducto", nativeQuery = true) void actualizarCantidadBodega(@Param("idBodega") Integer idBodega, @Param("idProducto") Integer idProducto, @Param("cantidadRecepcion") Integer cantidadRecepcion); // 6. Actualizar el estado de la Orden de Compra a 'ENTREGADA' @Transactional @Query(value = "UPDATE ORDENCOMPRAS SET ESTADO = 'ENTREGADA' WHERE ID_ORDENCOMPRAS = :idOrdencompra", nativeQuery = true) void actualizarEstadoEntregada(@Param("idOrdencompra") Integer idOrdencompra);</pre>
Después de los 30 segundos	T3	


		
	T4	

- ¿El componente que implementa RF10 debió esperar a que terminara la ejecución de la consulta RFC6 para poder registrar el ingreso de productos? Si el componente implemente el RF10 debió esperar a que terminara la ejecución de la consulta RFC6 como el nivel de aislamiento configurado en RFC6 es SERIALIZABLE. Esto se debe a la característica principal del nivel de aislamiento serializable ya que este nivel asegura que las transacciones se ejecuten de forma que parezcan haber sido ejecutadas secuencialmente, sin interferencias. Esto significa que la consulta RFC6 bloquea la tabla o las filas necesarias hasta que termine de ejecutarse, lo que impide que RF10 realice cambios hasta que RFC6 haya finalizado. Por lo tanto, RF10 tendría que esperar a que RFC6 libere los recursos bloqueados antes de poder realizar cualquier modificación en la base de datos.
- Resultado RFC6, ¿Apareció el documento de Ingreso de producto realizado al ejecutar el componente que implementa RF10 de manera simultánea? No, el documento de ingreso de producto realizado por RF10 no debería aparecer en los resultados de RFC6 si RF10 se ejecuta mientras RFC6 todavía está en progreso. Esto se debe a que RFC6, al ejecutarse con el nivel de aislamiento SERIALIZABLE, lee los datos tal como estaban al comienzo de la transacción y no ve los cambios realizados por RF10 mientras está en progreso.

Caso de Concurrency 2:

- Linea de tiempo

RFC7 – read committed	TIEMPO S	RF10
<p>Ejecutar consulta RFC7</p> <pre> @Service public class RecuperoProductoService { @Autowired private RecuperoProductoRepository recuperProductoRepository; @Transactional(isolation = Isolation.READ_COMMITTED) public List<Producto> recuperarProductosPorIdIngresos(Integer idIngresos, Integer idIngresos2) { try { // Temporizador de 30 segundos para simular la espera de concurrencia Thread.sleep(30000); // Llamada al método del repositorio return recuperProductoRepository.buscarProductosPorIdIngresos(idIngresos, idIngresos2); } catch (InterruptedException e) { System.out.println("Temporizador fue interrumpido: " + e.getMessage()); throw new RuntimeException("Temporizador fue interrumpido", e); } } } </pre>	T1	

	T2	<p>Ejecutar query rf10</p> <pre>// 1. Ejecutar Inserción de Producto @ modifying @ functional @ queryvalue = "DELETE FROM RECEPCIONPRODUCTO WHERE ID_RECEPCION = :idRecepcion", nativeQuery = true void actualizarRecepcionProducto(@Param("idRecepcion") Integer idRecepcion); // 2. Ejecutar Producto de la Botiga @ modifying @ functional @ queryvalue = "DELETE FROM BOTIGA_PRODUCTO WHERE ID_BOTIGA = :idBotiga AND ID_PRODUCTO = :idProducto", nativeQuery = true void eliminarProductoBotiga(@Param("idBotiga") Integer idBotiga, @Param("idProducto") Integer idProducto); // 3. Actualizar el estado de la Orden de Compra a 'Anulada' @ modifying @ functional @ queryvalue = "UPDATE ORDENCOPPA SET ESTADO = 'Anulada' WHERE ID_COPPA = :idCoppa", nativeQuery = true void actualizarEstadoOrdenCompra(@Param("idCoppa") Integer idCoppa); // 4. Insertar Recepción de Producto @ modifying @ functional @ queryvalue = "INSERT INTO RECEPCIONPRODUCTO (ID_RECEPCION, FECHA_RECEPCION, CANTIDAD_RECEPCION, CANTIDAD_TOTAL, ID_BOTIGA, ID_PRODUCTO) = " + "VALUES (:idRecepcion, SYSDATE, :cantidadRecepcion, :cantidadTotal, :idBotiga, :idProducto)", nativeQuery = true void insertarRecepcionProducto(@Param("idRecepcion") Integer idRecepcion, @Param("cantidadRecepcion") Integer cantidadRecepcion, @Param("cantidadTotal") Double cantidadTotal, @Param("idBotiga") Integer idBotiga, @Param("idProducto") Integer idProducto); // 5. Actualizar la cantidad en Botiga @ modifying @ functional @ queryvalue = "UPDATE BOTIGA_PRODUCTO SET CANTIDAD = CANTIDAD - :cantidadRecepcion WHERE ID_BOTIGA = :idBotiga AND ID_PRODUCTO = :idProducto", nativeQuery = true void actualizarCantidadBotiga(@Param("idBotiga") Integer idBotiga, @Param("idProducto") Integer idProducto, @Param("cantidadRecepcion") Integer cantidadRecepcion); // 6. Actualizar el estado de la Orden de Compra a 'ENTREGADA' @ modifying @ functional @ queryvalue = "UPDATE ORDENCOPPA SET ESTADO = 'ENTREGADA' WHERE ID_COPPA = :idCoppa", nativeQuery = true void actualizarEstadoOrdenCompra(@Param("idCoppa") Integer idCoppa);</pre>
	T3	
	T4	

- ¿El componente que implementa RF10 debió esperar a que terminara la ejecución de la consulta RFC7 para poder registrar el ingreso de productos? No, Cuando RFC7 (que es RFC6 configurado con READ COMMITTED) está en ejecución, RF10 no tiene que esperar a que RFC7 termine para realizar sus modificaciones. El nivel de aislamiento READ COMMITTED solo bloquea las filas que están siendo leídas por RFC7 en ese momento, pero no impide que otras transacciones (como RF10) realicen cambios en la base de datos.
- ¿Apareció el documento de Ingreso de producto realizado al ejecutar el componente que implementa RF10 de manera simultánea? Existen dos escenarios posibles: Si RF10 modifica las filas antes de que RFC7 las lea: RFC7 mostrará el documento de ingreso de producto realizado por RF10. Esto se debe a que RFC7 ve los datos más recientes comprometidos por otras transacciones, incluso si esos datos se modifican durante la ejecución de RFC7. Si RF10 modifica las filas después de que RFC7 las haya leído: RFC7 no mostrará el documento de ingreso de producto realizado por RF10, ya que RFC7 ya habrá leído los datos antes de que se realicen las modificaciones.