

Nicolás Ballén-202310273  
María Juliana Ballesteros-202313216  
Mauricio Martínez-202314461

# Proyecto SuperAndes

## Entrega 2-Aislamientos

### Grupo C7

#### Contexto

Nos vimos en la tarea de desarrollar una base de datos para la cadena de supermercados SuperAndes. Tras la lectura del informe, nuestro grupo identificó las siguientes entidades que componen el comportamiento y registros del supermercado:

- **Ciudad:** Comprende la ciudad en la que tiene actividad el supermercado.
- **Sucursal:** Comprende las diferentes sucursales del supermercado. Cada sucursal se ubica en una ciudad, habiendo múltiples sucursales por ciudad. Registra su nombre, dirección, teléfono y tamaño.
- **Bodega:** Comprende las bodegas donde se guardan los productos del supermercado. Cada bodega corresponde a una sucursal, pudiendo estas tener múltiples bodegas. Registra su tamaño y nombre.
- **Producto:** Comprende los diferentes productos que se venden en el supermercado. Reconoce su precio de venta, de bodega, código de barras, nombre, presentación, cantidad, unidades de medida, fecha de vencimiento, peso y volumen. Los productos pueden guardarse en múltiples bodegas.
- **Almacenaje:** Comprende los datos de cómo se almacena X producto en X bodega. Esto reconoce tanto el producto como la bodega en la que se guarda, la cantidad/capacidad de dicho producto en la misma, su costo promedio, su nivel mínimo de reorden (cantidad mínima de unidades que debe tener la bodega antes de solicitar más) y su cantidad.
- **Categoría:** Comprende los tipos de productos que hay en la sucursal. A cada producto le corresponde una categoría, habiendo varios productos por categoría. Esta reconoce una descripción y características de esta.
- **Proveedor:** Comprende a los diferentes proveedores que suministran al supermercado. Cada proveedor puede ofrecer diferentes productos, así como un producto puede ser ofrecido por varios proveedores. Comprenden su NIT, dirección, contacto y teléfono de contacto.
- **Orden de Compra:** Comprende las solicitudes que una sucursal hace por una bodega a un proveedor para comprar uno o más productos. Esta entiende los diferentes productos a comprar, la cantidad, un precio acordado por unidad, una fecha esperada para su recepción, una fecha de llegada y una de creación de la orden. Toda orden corresponde a un proveedor.
- **NUEVO: Registro de Ingresos:** Dados los nuevos requerimientos de esta entrega, se retomó la idea original del registro de ingreso, el cual resguarda información de la compra entregada, la fecha en la que se entregó, en qué bodega de la sucursal se guardó, y finalmente los productos que fueron ingresados.

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

## Resumen de Cambios Realizados – Entrega 2:

- Todos los requerimientos habían fallado en Postman pero funcionado en SQL Developer. Ahora ya funcionan todo correctamente en Postman.
- Las bodegas ya no pueden borrarse si tienen productos dentro de sí o su sucursal espera una orden de compra, la cual podría recibir.
- La creación de la orden ya recibe id de productos, precios y cantidades acordes, registrándolo en la tabla de ProductosOrden. En caso de que un producto escogido no pertenezca al proveedor seleccionado, se descartará, siendo el caso en el que se descartan todos donde no se crea orden alguna. La orden ahora no tendrá fecha de llegada ni bodega destino (ahora sucursal) dada la implementación de la tabla Registros.
- Implementación de la tabla Registros para el ingreso de productos de acuerdo a los requerimientos RFC6-7 y RF10. El último solo permite el ingreso en bodegas correspondientes que nunca han recibido el producto (asumimos arbitrariamente capacidad para el mismo) o que ya recibieron antes y tienen espacio suficiente.

## Modelo Relacional / Normalización

A continuación se presentarán las tablas/relación del modelo resultante, su llave candidata y por último un recorrido de como estas cumplen las formas normalizadas des 1FN a FN-BC.

### Ciudades:

iD	Nombre
PK	NN,ND

Solo conserva un iD y nombre que no puede ser nulo y asumimos que no puede repetirse (en Colombia no hay ciudades con el mismo nombre)

- Llave candidata: iD
- 1FN: No tenemos atributos multivalor. Toda ciudad tiene un único id y nombre
- 2FN: La llave candidata es única, así que nombre depende de esta en su totalidad.
- 3FN: Como solo hay 1 atributo no primo (nombre), es imposible incumplir esta forma.
- BC: Hay una única llave candidata que es el único primo, así que no hay otro primo del que depender.

### Sucursales:

iD	Nombre	Tamaño	Ciudad	Direccion	Telefono
PK	NN, ND	NN	NN, FK-Ciudades	NN	NN

Para cumplir la asociación uno a muchos de ciudad a sucursal, se toma un atributo ciudad en sucursales que no puede ser nulo, pero puede repetirse, de tal forma que una ciudad tenga varias sucursales asignadas a una única ciudad.

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

- Llave candidata: iD
- 1FN: No tenemos atributos multivalor. Todo atributo es atómico (no contemplamos múltiples teléfonos).
- 2FN: iD es el único primo, por lo que todos los demás atributos dependen en su totalidad de esta (sin parcialidad).
- 3FN: Ningún otro atributo puede determinar a los demás (solo iD), por lo que no podemos conocer ninguno de los atributos no primos sin el único primo (iD).
- BC: Hay una única llave candidata que es el único primo, así que no hay otro primo del que depender.

### Bodegas

iD	Nombre	Tamaño	Sucursal
PK	NN	NN	NN, FK-Sucursales

Para cumplir con la asociación entre bodega y sucursal (muchos a uno), las bodegas tienen un atributo de sucursal que no puede ser nulo, más se puede repetir. De esta forma la bodega se asigna a una única sucursal, y una sucursal puede tener varias bodegas a su nombre.

- Llave candidata: iD
- 1FN: No tenemos atributos multivalor. Todo atributo es atómico.
- 2FN: iD es el único primo, por lo que todos los demás atributos dependen en su totalidad de esta (sin parcialidad).
- 3FN: Ningún otro atributo puede determinar a los demás (solo iD), por lo que no podemos conocer ninguno de los atributos no primos sin el único primo (iD).
- BC: Hay una única llave candidata que es el único primo, así que no hay otro primo del que depender.

### Productos

Cod Barra s	No mbr e	Costo en Bodega	Precio Unitario	Presen tación	CantidadPre sentacion	Unidad Medida	Peso	Volumen	Fecha de Expiracion	Categ oría
PK	NN	NN	NN	NN	NN	NN, CK	NN	NN	NN	NN, FK- Categ orias

En el caso de producto, cumplimos su relación muchos a uno con categoría con un atributo de categoría que no puede ser nulo pero si repetible, habiendo así varias productos con una única categoría, la cual puede tener varias productos.

- Llave candidata: Código de Barras
- 1FN: No tenemos atributos multivalor. Todo atributo es atómico.
- 2FN: Código de Barras es el único primo, por lo que todos los demás atributos dependen en su totalidad de esta (sin parcialidad).

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

- 3FN: Ningún otro atributo puede determinar a los demás (solo el código), por lo que no podemos conocer ninguno de los atributos no primos sin el único primo.
- BC: Hay una única llave candidata que es el único primo, así que no hay otro primo del que depender.

### Almacenajes

Bodega	Producto	Capacidad	Costo Promedio	Nivel Mínimo
PK, FK-Bodegas	PK, FK-Productos	NN	NN	NN

Esta relación sirve para comprender los muchos productos que pueden guardarse en múltiples bodegas, teniendo así tanto la llave de bodega como la de producto

- Llave candidata: (Bodega, Producto)
- 1FN: No tenemos atributos multivalor. Todo atributo es atómico.
- 2FN: En este caso se necesita si o se tanto la bodega como el producto para definir los demás atributos. Dos bodegas pueden tener el mismo producto, pero la capacidad, costo, y nivel mínimo que tienen en cada pueden ser distintos. De igual forma una bodega tendrán varios productos, pero los atributos de estos dentro de ella no serán necesariamente iguales. Por ello se necesitan ambos para ser si o si la llave candidata.
- 3FN: Los datos ajenos a la llave candidata corresponden únicamente al producto en la bodega correspondiente, por lo que estos no pueden determinarse entre sí. No hay no primos que determinen no primos.
- BC: La llave candidata es única, así como bodega no puede determinar el producto en cuestión (la bodega contiene muchos productos) ni viceversa (el producto se puede guardar en muchas bodegas). Por ende ningún primo determina otro primo.

### Proveedor

NIT	Nombre	Contacto	Tel. Contacto
PK	NN	NN	NN

- Llave candidata: NIT
- 1FN: No tenemos atributos multivalor. Todo atributo es atómico.
- 2FN: El NIT es la llave candidata en su totalidad, y al ser la única significa que el resto de atributos (no primos) son determinados en su totalidad por esta.
- 3FN: El nombre no determina el contacto ni su teléfono. Podría existir que el número de teléfono no se repitiese y con este se determinaría el resto, más este no es el caso.
- BC: La llave candidata es única, así como el único primo. Por ende no hay dependencia alguna entre primos.

### Categorías

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

Código	Nombre	Descripción	Características
PK	NN, ND	NN	NN

- Llave candidata: Código
- 1FN: No tenemos atributos multivalor. Todo atributo es atómico (las descripciones y características son un único texto, seguramente de longitud mayor).
- 2FN: El código es la llave candidata en su totalidad, y al ser la única significa que el resto de atributos (no primos) son determinados en su totalidad por esta.
- 3FN: El resto de atributos no se determinan entre si.
- BC: La llave candidata es única, así como el único primo. Por ende no hay dependencia alguna entre primos.

### Ofertas

Proveedor	Producto
PK, FK-Bodegas	PK, FK-Productos

Esta relación sirve a la asociación muchos a muchos entre proveedores y productos, definiendo los productos que ofrecen los diferentes proveedores.

- Llave candidata: (Proveedor, Producto)
- 1FN: No tenemos atributos multivalor. Todo atributo es atómico (se usan las llaves primarias de cada uno).
- 2FN: No hay atributos no-primos como tal, así que no puede haber dependencias.
- 3FN: No hay atributos no-primos como tal, así que no puede haber dependencias entre estos
- BC: Los dos atributos componen la única llave candidata, así que no pueden determinarse entre sí. De por sí solo se tiene el dato de ambos, los cuales se repiten para los múltiples proveedores con múltiples productos.

### Ordenes

iD	Proveedor	FechaEstimada	FechaCreacion	Estado
PK	NN, FK-Proovedores	NN	NN	NN, CK

En este caso destaco la relación muchos a uno de orden a proveedor, teniendo la orden un atributo repetible Proveedor que hace referencia al mismo en múltiples órdenes. Adicionalmente, su estado se marca con CK dados los determinados estados que puede tener la orden desde VIGENTE hasta ENTREGADA.

- Llave candidata: iD
- 1FN: No tenemos atributos multivalor. Todo atributo es atómico (se usan las llaves primarias en la foránea).
- 2FN: Como la llave candidata es atómica, si o si todos los demás atributos dependen de esta en su totalidad.

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

- 3FN: Ningún otro atributo que no sea el iD puede determinar al resto.
- BC: La llave candidata, así como los atributos primos, son solo uno. Ningún primo determina a otro.

### ProductosOrden

Orden	Producto	Cantidad	Precio Acordado
PK, FK-Orden	PK, FK-Productos	NN	NN

Sirve a la relación muchos a muchos de orden y productos.

- Llave candidata: (Orden, Producto)
- 1FN: No tenemos atributos multivalor. Todo atributo es atómico (se usan las llaves primarias en la foránea).
- 2FN: Necesitamos tanto de la orden como del producto para saber cuánto del producto se ordenó y cuál fue el precio acordado por unidad. Sin producto no sabemos de qué producto en específico hablamos (la orden tiene puede tener muchos), y sin orden no sabremos cuánto del producto se pidió y por cuánto (dependiendo del proveedor en la orden los datos serán diferentes). Por ende, se necesita la llave candidata en su totalidad.
- 3FN: La cantidad del producto no define el precio acordado, sino el producto y la orden. Ningún no primo define otro no primo.
- BC: La llave candidata es única. Ningún primo determina a otro (la orden no define al producto en específico ni le producto a la orden)

### Registros

Orden	FechaIngreso	Bodega
PK, FK (ordenes. Id)	NN	NN, FK (bodegas.id)

Mantiene una relación uno a uno con Ordenes, así como Muchas a uno con Bodegas.

- Llave candidata: (Orden)
- 1FN: No tenemos atributos multivalor. Todo atributo es atómico (se usan las llaves primarias en la foránea).
- 2FN: La llave primaria es de un solo atributo, así que de ella depende obtener la fecha de ingreso y la bodega.
- 3FN: No hay forma de definir la bodega con la fecha ni viceversa, siendo que la bodega recibe varias ordenes un mismo día, así como en una misma fecha se reciben ordenes en diferentes bodegas.
- BC: La llave candidata es única, así como el único primo.

Nicolás Ballén-202310273  
María Juliana Ballesteros-202313216  
Mauricio Martinez-202314461

## Documentacion Requerimientos:

### RF1 – Crear una ciudad

Este requerimiento permite crear una ciudad en la base de datos. El método insertarCiudad ejecuta una consulta SQL en la cual se inserta en la tabla “Ciudades” una nueva ciudad con los parámetros id y nombre. Para el id se utiliza un generador secuencial (paso.nextval) para asegurar que cada registro tenga un id único. El parámetro “Nombre” representa el nombre de la ciudad que se va a insertar y es pasado dinámicamente a la consulta.

```
@Modifying
@Transactional
@Query(value = "INSERT INTO ciudades (id, nombre) VALUES ( paso.nextval , :nombre)", nativeQuery = true)
void insertarCiudad(@Param("nombre") String nombre);
```

Para verificar que el Api permita insertar una nueva ciudad en la base de datos correctamente, se utiliza una solicitud de tipo POST dirigida a la URL “<http://localhost:8080/superandesC7/ciudades/new/save>”. En el cuerpo de la solicitud se envía un JSON con la estructura `{"nombre": "nombreCiudad"}`. La respuesta esperada debe devolver el código de estado “201” y un JSON con la nueva ciudad, con el id generado y el nombre de la ciudad.

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** <http://localhost:8080/superandesC7/ciudades/new/save>
- Body (raw JSON):**

```
1 {
2   "nombre": "Alaska"
3 }
```
- Response Status:** 201 Created
- Response Body:** Ciudad creada exitosamente

Adicionalmente, dado lo simple del registro de ciudad en un único atributo, se delimito el ingreso de ciudades con exactamente el mismo nombre

Nicolás Ballén-202310273  
María Juliana Ballesteros-202313216  
Mauricio Martínez-202314461

The screenshot shows a Postman interface with a POST request to `http://localhost:8080/superandesC7/ciudades/new/save`. The Body tab is selected, showing the raw JSON input: `{ "nombre": "Alaska" }`. The response status is `409 Conflict` with the message `La ciudad ya existe`.

## RF2 – Crear una sucursal

Este requerimiento permite crear una sucursal en la base de datos. El método `insertarSucursal` ejecuta una consulta SQL que recibe por parámetro el id de la sucursal para el cual se utiliza un generador secuencial, el nombre de la sucursal, el tamaño de la sucursal, la ciudad en la que se encuentra la sucursal la cual se referencia con el id de la ciudad, el teléfono de contacto y la dirección.

```
@Modifying
@Transactional
@Query【value = "INSERT INTO sucursales (id, nombre, tamaño, ciudad, telefono, direccion)
VALUES ( :paso.nextval, :nombre, :ciudad, :telefono, :direccion)", nativeQuery = true】
void insertarSucursal(@Param("nombre") String nombre, @Param("tamaño") Long tamaño, @Param("ciudad") Long ciudad,
@Param("telefono") Long telefono, @Param("direccion") String direccion);
```

Para verificar que el API permita insertar una nueva sucursal en la base de datos correctamente, se utiliza una solicitud de tipo POST dirigida a la URL “<http://localhost:8080/superandesC7/sucursales/new/save>”. En el cuerpo de la solicitud se envía un JSON con la estructura `{"nombre": "nombreSucursal", "tamaño": "tamañoSucursal", "direccion": "direcciónSucursal", "telefono": "telefonoContacto", "ciudad": {"id": ciudadId}}`. La respuesta esperada debe devolver el código de estado “201” y un JSON con la nueva sucursal y el id generado.

The screenshot shows a Postman interface with a POST request to `http://localhost:8080/superandesC7/sucursales/new/save`. The Body tab is selected, showing the raw JSON input: `{ "nombre": "sucursal1", "tamaño": 29, "direccion": "Calle 127", "telefono": 123, "ciudad": {"id": 1} }`. The response status is `201 Created` with the message `Sucursal creado exitosamente`.

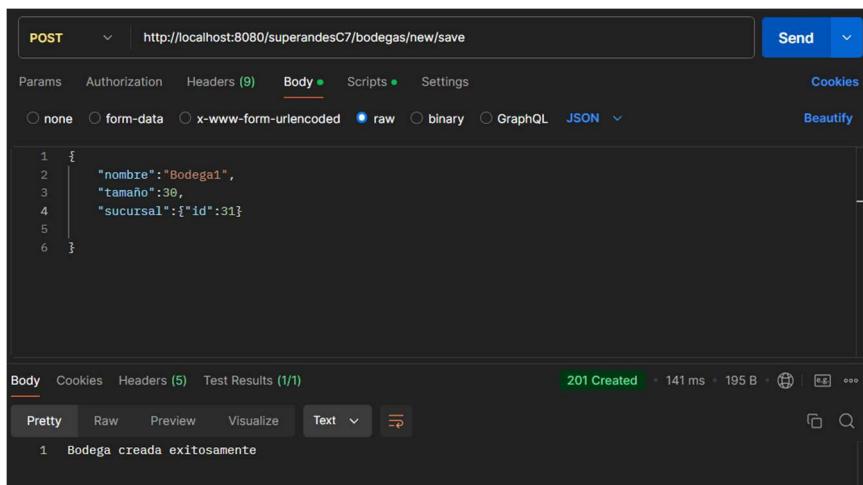
Nicolás Ballén-202310273  
María Juliana Ballesteros-202313216  
Mauricio Martinez-202314461

### RF3 – Crear y borrar una bodega

Este requerimiento permite crear y borrar una bodega de la base de datos. En el método insertarBodega se ejecuta una secuencia SQL que inserta una bodega nueva en la tabla “Bodegas”, recibe como parámetro el id de la bodega que se genera con un generador secuencial, el nombre de la bodega, el tamaño de la bodega y el identificador de la sucursal a la que pertenece la bodega.

```
@Modifying
@Transactional
@Query(value = "INSERT INTO bodegas (id, nombre, tamaño, sucursal)
VALUES (:paso.nextval, :nombre, :tamaño, :sucursal)", nativeQuery = true)
void insertarBodega(@Param("nombre") String nombre, @Param("tamaño") Long tamaño, @Param("sucursal") Long sucursal);
```

Para verificar que el Api permita insertar una nueva bodega en la base de datos correctamente, se utiliza una solicitud de tipo POST dirigida a la URL “<http://localhost:8080/superandesC7/bodegas/new/save>”. En el cuerpo de la solicitud se envía un JSON con la estructura `{"nombre": "nombreBodega", "tamaño": "tamañoBodega", "sucursal": {"id": sucursalId}}`. La respuesta esperada debe devolver el código de estado “201” y un JSON con la nueva bodega y el id generado.



Para eliminar una bodega se tienen dos métodos. El primero eliminarBodega, elimina una bodega dado un id, el método ejecuta una sentencia SQL que elimina la bodega de la tabla “bodegas” que tenga un id que entra por parámetro. El segundo elimina las bodegas de una sucursal, el método ejecuta una sentencia SQL que elimina las bodegas que pertenezcan a una sucursal de la tabla “Bodegas”.

```
@Modifying
@Transactional
@Query(value = "DELETE FROM bodegas WHERE id = :id", nativeQuery = true)
void eliminarBodega(@Param("id") Long id);

@Modifying
@Transactional
@Query(value = "DELETE FROM bodegas WHERE sucursal = :sucursal", nativeQuery = true)
void eliminarBodegaPorSucursal(@Param("sucursal") Long sucursal);
```

Para verificar que el Api permita borrar una bodega existente en la base de datos correctamente, se utiliza una solicitud de tipo DEL dirigida a la URL “[http://localhost:8080/superandesC7/bodegas/{{id\\_bodega}}/delete](http://localhost:8080/superandesC7/bodegas/{{id_bodega}}/delete)”. En el cuerpo de la solicitud se envía un JSON con la estructura `{"bodega": {"id": bodegaId}}`. La respuesta esperada debe devolver el código de estado “204”.

Nicolás Ballén-202310273  
María Juliana Ballesteros-202313216  
Mauricio Martínez-202314461

The screenshot shows the Postman interface with the following details:

- Method:** DELETE
- URL:** http://localhost:8080/superandesC7/bodegas/42/delete
- Params:** Authorization, Headers (7), Body, Scripts •, Settings
- Query Params:** Key, Value, Description, Bulk Edit
- Body:** Key, Value, Description
- Headers:** (5)
- Test Results:** (0/1)
- Status:** 200 OK, 99 ms, 193 B
- Pretty:** Selected
- Raw, Preview, Visualize:** Options
- Text:** Text dropdown with a copy icon
- Content:** 1. Bodega eliminada exitosamente

**NUEVO** Sin embargo, la bodega no se borrará si tiene algo almacenado o si su sucursal esta en la espera de una orden de compra, siendo que podría llegar a recibirla.

DELETE <http://localhost:8080/superandesC7/bodegas/1/delete> Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description	...	

Body Cookies Headers (4) Test Results (0/1) 500 Internal Server Error • 86 ms • 275 B • [e.g.](#) ...

Pretty Raw Preview Visualize Text

1 No se puede eliminar la bodega-sucursal dueña tienes ordenes pendientes por recibir o aún hay inventario en la bodega.

## **RF4 – Crear y actualizar proveedores**

Para crear un proveedor se crea el método “insertarProveedor” el cual ejecuta una sentencia SQL que se encarga de insertar en la tabla “Proveedores” el nuevo proveedor. Se reciben por parámetro el NIT, el nombre, el contacto, el teléfono de contacto y la dirección del proveedor. En este caso el NIT es la PK y no es generada sino que es pasado dinámicamente a la consulta.

```
@Modifying  
@Transactional  
@Query(value = "INSERT INTO proveedores (nit, nombre, contacto, tel_contacto, direccion)  
VALUES (:nit , :nombre, :contacto, :tel_contacto, :direccion)", nativeQuery = true)  
void insertarProveedor(@Param("nit") Long nit, @Param("nombre") String nombre, @Param("contacto") String contacto,  
@Param("tel_contacto") Long tel_contacto, @Param("direccion") String direccion);
```

Para verificar que el API permite insertar un nuevo proveedor en la base de datos correctamente, se utiliza una

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

solicitud de tipo POST dirigida a la URL “<http://localhost:8080/superandesC7/proveedores/new/save>”. En el cuerpo de la solicitud se envía un JSON con la estructura `{"nit": nitProveedor, "nombre": "nombreProveedor", "direccion": "direccionProveedor", "contacto": "nombreContacto", "tel_contacto": telefonoContacto}`. La respuesta esperada debe devolver el código de estado “201” y un JSON con el nuevo proveedor.

The screenshot shows a Postman interface with a POST request to `http://localhost:8080/superandesC7/proveedores/new/save`. The request body is set to raw JSON:

```
1 {
2     "nit": 12553,
3     "nombre": "Chelet",
4     "direccion": "Ce 154",
5     "contacto": "Ana",
6     "tel_contacto": 555
7 }
```

The response tab shows a status of 201 and the message "Proveedor creado exitosamente".

Para actualizar un proveedor, el método `actualizarProveedor` ejecuta una sentencia UPDATE para actualizar los datos de un proveedor en la tabla “proveedores” estableciendo nuevos valores para los campos nombre, contacto, `tel_contacto` y `direccion` donde el `nit` coincide con el valor proporcionado.

```
@Modifying
@Transactional
@Query(value = "UPDATE proveedores SET nombre = :nombre, contacto = :contacto, tel_contacto = :tel_contacto, direccion = :direccion WHERE nit = :nit", nativeQuery = true)
void actualizarProveedor(@Param("nit") Long nit, @Param("nombre") String nombre, @Param("contacto") String contacto, @Param("tel_contacto") Long tel_contacto, @Param("direccion") String direccion);
```

Para verificar que el API permite actualizar proveedor existente en la base de datos correctamente, se utiliza una solicitud de tipo PUT dirigida a la URL “[http://localhost:8080/superandesC7/proveedores/{proveedor\\_creado}/edit/save](http://localhost:8080/superandesC7/proveedores/{proveedor_creado}/edit/save)”. En el cuerpo de la solicitud se envía un JSON con la estructura `{"nit": nitProveedor, "nombre": "nombreProveedor", "direccion": "direccionProveedor", "contacto": "nombreContacto", "tel_contacto": telefonoContacto}`. La respuesta esperada debe devolver el código de estado “200” y un JSON con los datos del proveedor actualizados.

Nicolás Ballén-202310273  
María Juliana Ballesteros-202313216  
Mauricio Martínez-202314461

The screenshot shows a Postman request for a PUT endpoint at <http://localhost:8080/superandesC7/proveedores/12553/edit/save>. The request body contains the following JSON:

```
1 {  
2   "nit": 12553,  
3   "nombre": "Bimbo2",  
4   "direccion": "Calle 144",  
5   "contacto": "juan",  
6   "tel_contacto": 736  
7 }
```

The response status is 200 OK, and the message is "Proveedor actualizado exitosamente".

## RF5 – Crear y leer una categoría de producto

Para crear una categoría se usa el método insertarCategoria el cual ejecuta una sentencia SQL que inserta en la tabla “categorias” una nueva categoría. Con un código asignado automáticamente y recibe los parámetros nombre, descripción y características del producto.

```
@Modifying  
@Transactional  
@Query["value = "INSERT INTO categorias (codigo, nombre, descripcion, caracteristicas)  
VALUES (:paso.nextval , :nombre, :descripcion, :caracteristicas)", nativeQuery = true]  
void insertarCategoria(@Param("nombre") String nombre, @Param("descripcion") String descripcion, @Param("caracteristicas") String caracteristicas);
```

Para verificar que el API permite crear una nueva categoría en la base de datos correctamente, se utiliza una solicitud de tipo POST dirigida a la URL “<http://localhost:8080/superandesC7/categorias/new/save>”. En el cuerpo de la solicitud se envía un JSON con la estructura `{"nombre": "nombreCategoria", "descripcion": "descripcionCategoria", "caracteristicas": "caracteristicasCategoria"}`. La respuesta esperada debe devolver el código de estado “201” y un JSON con la nueva categoría creada.

The screenshot shows a Postman request for a POST endpoint at <http://localhost:8080/superandesC7/categorias/new/save>. The request body contains the following JSON:

```
1 {  
2   "nombre": "Congelados",  
3   "descripcion": "Productos congelados",  
4   "caracteristicas": "Se deben mantener congelados"  
5 }  
6 }
```

The response status is 201 Created, and the message is "Categoria creada exitosamente".

Una categoría se puede leer por nombre o código. El método darCategoriaPorNombre obtiene una categoría por su nombre, con una consulta de SQL selecciona todos los registros de la tabla “categorias” donde el campo nombre es igual a el valor proporcionado. El método darCategoria obtiene una categoría por su código, hace lo mismo que darCategoriaPorNombre pero selecciona los registros de la tabla donde el campo código sea igual al valor proporcionado.

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

```
@Query(value = "SELECT * FROM categorias WHERE codigo = :codigo", nativeQuery = true)
Categoria darCategoria(@Param("codigo") Long codigo);

@Query(value = "SELECT * FROM categorias WHERE nombre = :nombre", nativeQuery = true)
Collection<Categoria> darCategoriaPorNombre(@Param("nombre") String nombre);
```

Para verificar que la API permita leer una categoría por nombre, se utiliza una solicitud de tipo GET dirigida a la URL “<http://localhost:8080/superandesC7/categorias/consulta?nombre=Daltfresh>”. El resultado esperado es un JSON con las categorías que tienen el nombre dado.

GET http://localhost:8080/superandesC7/categorias/consulta?nombre=Daltfresh

Params • Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
nombre	Daltfresh	

Body Cookies Headers (5) Test Results 200 OK

Pretty Raw Preview Visualize JSON

```
1 [ 
2   {
3     "codigo": 2,
4     "nombre": "Daltfresh",
5     "descripcion": "Emerald Green Neutral pH Anti-Bacterial Hand",
6     "caracteristicas": "Benefon"
7   }
8 ]
```

Para verificar que la API permita leer una categoría por código, se utiliza una solicitud de tipo GET dirigida a la URL “<http://localhost:8080/superandesC7/categorias/2>”. El resultado esperado es un JSON con las categorías que tienen el código dado con el mismo resultado.

GET http://localhost:8080/superandesC7/categorias/2

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
Key	2	

Body Cookies Headers (5) Test Results 200 OK

Pretty Raw Preview Visualize JSON

```
1 { 
2   "codigo": 2,
3   "nombre": "Daltfresh",
4   "descripcion": "Emerald Green Neutral pH Anti-Bacterial Hand",
5   "caracteristicas": "Benefon"
6 }
```

## RF6 – Crear, Leer y Actualizar un producto

Para crear un producto se usa el método insertarProducto el cual ejecuta una sentencia SQL que inserta un nuevo producto en la tabla “productos”. El código de barras es generado automáticamente y recibe por parámetro el identificador de la categoría a la que pertenece el producto, nombre, costo\_bodega, precio\_unitario, presentación, peso, volumen, unidad\_medida, cantidad\_presentacion, fecha\_vencimiento.

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

```
@Modifying
@Transactional
@Query(value = "INSERT INTO productos (cod_barra, nombre, costo_bodega, precio_unitario, presentacion, peso, volumen, unidad_medida, cantidad_presentacion, fecha_vencimiento, categoria)
VALUES(:paso.nextval, :nombre, :costo_bodega, :precio_unitario, :presentacion, :peso, :volumen, :unidad_medida, :cantidad_presentacion, :fecha_vencimiento, :categoria)", nativeQuery = true)
void insertarProducto(@Param("nombre") String nombre, @Param("costo_bodega") Long costo_bodega, @Param("precio_unitario") Long precio_unitario,
@Param("presentacion") String presentacion, @Param("peso") Long peso, @Param("volumen") Long volumen, @Param("unidad_medida") String unidad_medida,
@Param("cantidad_presentacion") Long cantidad_presentacion, @Param("fecha_vencimiento") Date fecha_vencimiento, @Param("categoria") Long categoria);
```

Para verificar que el API permite crear un nuevo producto correctamente, se utiliza una solicitud de tipo POST dirigida a la URL “<http://localhost:8080/superandesC7/productos/new/save>”. En el cuerpo de la solicitud se envía un JSON con la estructura `{"nombre": "nombreProducto", "costo_bodega": costo, "precio_unitario": precioUnitario, "presentacion": "presentacionProducto", "cantidad_presentacion": cantidadPresentacion, "unidad_medida": "unidadesMedida", "peso": peso, "volumen": volumen, "fecha_vencimiento": "fechaVencimiento", "categoria": {"codigo": codigoCategoria}}`. La respuesta esperada debe devolver el código de estado “201” y un JSON con el producto nuevo.

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** <http://localhost:8080/superandesC7/productos/new/save>
- Body:** Raw JSON (selected)

```
1 {
2     "nombre": "Shampoo",
3     "costo_bodega": 20,
4     "precio_unitario": 20,
5     "presentacion": "Botella de 100ml",
6     "cantidad_presentacion": 100,
7     "unidad_medida": "ml",
8     "peso": 20,
9     "volumen": 20,
10    "fecha_vencimiento": "2025-10-11",
11    "categoria": {"codigo": 2}
```
- Headers:** (9)
- Response Status:** 201 Create
- Response Body:** Producto creado exitosamente

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

Para leer un producto se utiliza el método darProducto y darProductoPorNombre. El primero ejecuta una sentencia SQL que recibe por parámetro el código de barras de un producto, mientras que el nombre, selecciona todos los registros de la tabla “productos” donde el valor de código de barras/nombre coincide con el valor proporcionado.

```
@Query(value = "SELECT * FROM productos WHERE cod_barras= :cod_barras", nativeQuery=true)
Producto darProducto(@Param("cod_barras") int cod_barras);

@Query(value = "SELECT * FROM productos WHERE nombre= :nombre", nativeQuery=true)
Collection<Producto> darProductoPorNombre(@Param("nombre") String nombre);
```

Para verificar que el API permite leer un producto, se utiliza una solicitud de tipo GET. El resultado esperado debe devolver un JSON con el producto con código de barras igual al que se solicitó.

The screenshot shows two separate API calls in Postman:

- Request 1:** GET to `http://localhost:8080/superandesC7/productos/1`.
  - Params: None
  - Body: None
  - Headers: (7)
  - Test Results: None
- Request 2:** GET to `http://localhost:8080/superandesC7/productos/consultaNombre?nombre=Bitwolf`.
  - Params: None
  - Body: None
  - Headers: (5)
  - Test Results: 200

Both requests return a JSON response with the following structure:

```
1 [ { "cod_barras": 1, "nombre": "Bitwolf", "costo_bodega": 8, "precio_unitario": 375, "presentacion": "Ephippiorhynchus senegalensis", "cantidad_presentacion": 2, "unidad_medida": "SRR", "peso": 78626, "volumen": 78626, "fecha_vencimiento": "2023-03-15", "categoria": { "codigo": 1, "nombre": "Ventosanzap", "descripcion": "CEFTRIAXONE", "caracteristicas": "BenQ-Siemens" } }
```

Para actualizar un producto en la base de datos se usa el método actualizarProducto. La consulta SQL hace un UPDATE en la tabla “productos”, estableciendo nuevos valores para los campos nombre, costo\_bodega, precio\_unitario, presentacion, peso, volumen, unidad\_medida, cantidad\_presentacion, fecha\_vencimiento y categoria de un producto con código de barras igual al valor que entra por parámetro.

```
@Modifying
@Transactional
@Query(value = "UPDATE productos SET nombre=:nombre, costo_bodega=:costo_bodega, precio_unitario=:precio_unitario, presentacion=:presentacion, peso=:peso, volumen=:volumen, unidad_medida=:unidad_medida, cantidad_presentacion=:cantidad_presentacion, fecha_vencimiento=:fecha_vencimiento WHERE cod_barras=:cod_barras", nativeQuery=true)
void actualizarProducto(@Param("cod_barras") Long cod_barras, @Param("nombre") String nombre, @Param("costo_bodega") Long costo_bodega, @Param("precio_unitario") Double precio_unitario, @Param("presentacion") String presentacion, @Param("peso") Long peso, @Param("volumen") Long volumen, @Param("unidad_medida") String unidad_medida, @Param("cantidad_presentacion") Long cantidad_presentacion, @Param("fecha_vencimiento") Date fecha_vencimiento, @Param("categoria") Long categoria);
```

Para verificar que el API permite actualizar un producto, se utiliza una solicitud de tipo PUT dirigida a la URL `“http://localhost:8080/superandesC7/productos/{codigo}/edit/save”`. En el cuerpo de la solicitud se envía un JSON con la estructura `{"nombre": "nombreProducto", "costo_bodega": costo, "precio_unitario": precioUnitario, "presentacion": "presentacionProducto", "cantidad_presentacion": cantidadPresentacion, "unidad_medida": "unidadesMedida", "peso": peso, "volumen": volumen, "fecha_vencimiento": "fechaVencimiento", "categoria": {"codigo": codigoCategoria}}}`. El resultado esperado debe devolver el código de “200” y un JSON con el producto actualizado.

Nicolás Ballén-202310273  
 María Juliana Ballesteros-202313216  
 Mauricio Martínez-202314461

The screenshot shows a POSTMAN interface with a 'PUT' request to the URL `http://localhost:8080/superandesC7/productos/35/edit/save`. The 'Body' tab is selected, showing a JSON payload with product details:

```

1  {
2      "nombre": "Pasta",
3      "costo_bodega": 100,
4      "precio_unitario": 5,
5      "presentacion": "paquete de 14gr",
6      "cantidad_presentacion": 14,
7      "unidad_medida": "gr",
8      "peso": 100,
9      "volumen": 20,
10     "fecha_vencimiento": "2026-09-11",
11     "categoria": {"codigo": 1}
12

```

The response status is '200 OK' with the message 'Producto actualizado exitosamente'.

## RF7 – Crear una orden de compra para una sucursal

El método insertarOrden permite insertar una nueva orden en la base de datos, asignándole automáticamente un identificador único (id) y estableciendo su estado inicial como 'vigente'. La consulta SQL nativa inserta valores en la tabla ordenes para los campos fecha\_creacion, fecha\_estimada, proveedor y sucursal\_destino, que son proporcionados a través de los parámetros. Esto permite crear una nueva orden con las fechas de creación y estimada, así como con la referencia al proveedor y la sucursal de destino.

```

@Modifying
@Transactional
@Query【value = "INSERT INTO ordenes (id, estado, fecha_creacion, fecha_estimada, proveedor, sucursal_destino)
VALUES(:paso.nextval, 'vigente', :fecha_creacion, :fecha_estimada, :proveedor, :sucursal_destino)", nativeQuery = true】
void insertarOrden(@Param("fecha_creacion")Date fecha_creacion, @Param("fecha_estimada")Date fecha_estimada,
@Param("proveedor")Long proveedor, @Param("sucursal_destino")Long sucursal_destino);

```

Para verificar que el API permite crear una orden de compra en la base de datos, se utiliza una solicitud de tipo PUT dirigida a la URL "`http://localhost:8080/superandesC7/ordenes/new/save`". La respuesta esperada debe devolver el código de estado "201" y un JSON con la orden de compra creada.

A esto se le suma la solicitud de parámetros correspondientes a los productos, cantidades, y precios acordados de cada producto en forma de arreglos

```

@PostMapping("/ordenes/new/save")
public ResponseEntity<String> ordenGuardar(@RequestBody Orden norden, @RequestParam(required = true) String productos, @RequestParam(required = true) String precios, @Request
try{
    long[] id_productos= Arrays.stream(productos.split(",")).mapToLong(f -> Long.parseLong(f)).toArray();
    long[] val_precios= Arrays.stream(precios.split(",")).mapToLong(f -> Long.parseLong(f)).toArray();
    long[] val_cantidades= Arrays.stream(productos.split(",")).mapToLong(f -> Long.parseLong(f)).toArray();
    norden.setFechaCreacion(new Date(new java.util.Date().getTime()));
    if (norden.getFecha_estimada().compareTo(norden.getFecha_creacion())<0){
        return new ResponseEntity<>(body:"Fecha inválida, debe ser posterior a la actual", HttpStatus.CREATED);
    }
}

```

Estos arreglos se verifican y por cada producto se agregan a la tabla de ProductosOrden para concordar con la orden de compra estipulada. También se descartan los productos que no pertenecen al proveedor o que no los tiene, borrando la orden sin ningún producto está disponible

Nicolás Ballén-202310273  
María Juliana Ballesteros-202313216  
Mauricio Martínez-202314461

The screenshot shows a Postman interface with a POST request to the URL `http://localhost:8080/superandesC7/ordenes/new/save?productos=6,9&precios=10,10&cantidades=11,11`. The 'Query Params' section contains four entries: `productos` with value `6,9`, `precios` with value `10,10`, `cantidades` with value `11,11`, and an empty entry for `Key` with `Value` and `Description` fields. Below the request, the response details show a `201 Created` status with a response time of `199 ms` and a size of `273 B`. The response body contains the message: `1 Orden creada. Los siguientes productos no se incluyeron por falta de disponibilidad con el proveedor:`.

#### RF8 – Actualizar una orden de compra cambiando su estado a anulada

El método `actualizarOrdenAnulada` permite cambiar el estado de una orden en la base de datos a "anulada", siempre que su estado actual sea "vigente". La consulta SQL nativa actualiza el campo `estado` en la tabla `ordenes`, estableciéndolo en "anulada" únicamente si el id de la orden coincide con el valor proporcionado y si el estado actual es "vigente". Esto asegura que solo las órdenes activas puedan ser anuladas, preservando la lógica de negocio definida.

```
@Modifying
@Transactional
@Query(value = "UPDATE ordenes SET estado = 'anulada' WHERE id = :id AND estado = 'vigente' ", nativeQuery = true)
void actualizarOrdenAnulada(@Param("id") Long id);
```

Para verificar que el API actualice una orden de compra correctamente, se utiliza una solicitud de tipo PUT dirigida a la URL "`http://localhost:8080/superandesC7/ordenes/{id}/edit/anular/save`". La respuesta esperada debe devolver el código de estado "200" y un JSON con la orden de compra actualizada donde el estado es "Anulada".

The screenshot shows a Postman interface with a PUT request to the URL `http://localhost:8080/superandesC7/ordenes/44/edit/anular/save`. The 'Body' tab is selected, showing the option `none` selected. Below the request, the response details show a `200 OK` status. The response body contains the message: `1 Orden anulada exitosamente`.

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

Una orden previamente anulada o entregada no puede anularse, por lo que dará error y se le denotará al usuario:

PUT http://localhost:8080/superandesC7/ordenes/44/edit/anular/save

Params Authorization Headers (8) Body Scripts Settings

Body Cookies Headers (4) Test Results 500 Internal Server Error

Pretty Raw Preview Visualize Text This request does not have a body

1 Error al anular la orden-orden ya anulada

PUT http://localhost:8080/superandesC7/ordenes/39/edit/anular/save

Params Authorization Headers (8) Body Scripts Settings

Body Cookies Headers (4) Test Results 500 Internal Server Error

Pretty Raw Preview Visualize Text This request does not have a body

1 Error al anular la orden-orden entregada

## RF9 – Mostrar todas las ordenes de compra

El método darOrdene selecciona todas las ordenes de compra que se encuentran en la base de datos con toda su información.

```
@Query(value = "SELECT * FROM ordenes", nativeQuery=true )
Collection<Orden> darOrdenes();
```

Para verificar que el API permite obtener todas las ordenes de compra, se hace una solicitud de tipo GET dirigida a la URL “http://localhost:8080/superandesC7/ordenes”. El resultado debe devolver un JSON con todas las ordenes de compra con su información.

GET http://localhost:8080/superandesC7/ordenes

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK 167 ms 4.52 KB

Pretty Raw Preview Visualize JSON This request does not have a body

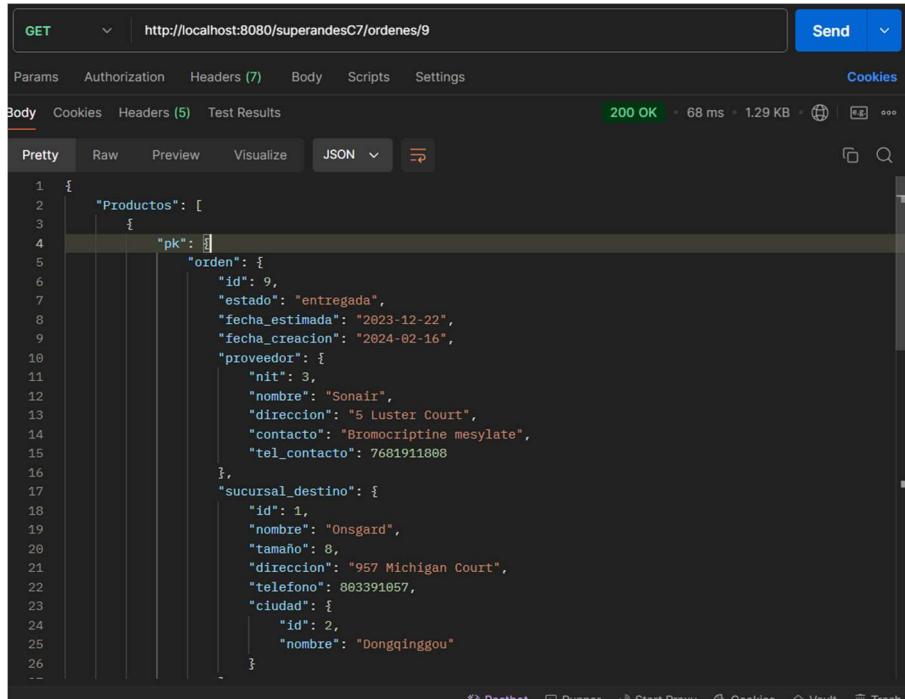
```
1 [ 2 { 3   "id": 1, 4   "estado": "entregada", 5   "fecha_estimada": "2024-01-08", 6   "fecha_creacion": "2024-05-28", 7   "proveedor": { 8     "nit": 4, 9     "nombre": "Flowdesk", 10    "direccion": "1453 Petterle Place", 11    "contacto": "Verapamil Hydrochloride", 12    "tel_contacto": 9385739373 13  }, 14  "sucursal_destino": { 15    "id": 8, 16    "nombre": "Shoshone", 17    "tamaño": 19549, 18  }}, 19 ]
```

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martinez-202314461

Adicionalmente agregamos la visualización de detalles de una orden, con lo cual podemos ver los productos asociados y su información agregando el id de la orden al path:



```
GET http://localhost:8080/superandesC7/ordenes/9
```

Body

```
1 {
2     "Productos": [
3         {
4             "pk": 9,
5                 "orden": {
6                     "id": 9,
7                         "estado": "entregada",
8                         "fecha_estimada": "2023-12-22",
9                         "fecha_creacion": "2024-02-16",
10                        "proveedor": {
11                            "nit": 3,
12                                "nombre": "Sonair",
13                                "direccion": "5 Luster Court",
14                                "contacto": "Bromocriptine mesylate",
15                                "tel_contacto": 7681911888
16                            },
17                            "sucursal_destino": {
18                                "id": 1,
19                                    "nombre": "Onsgard",
20                                    "tamaño": 8,
21                                    "direccion": "957 Michigan Court",
22                                    "telefono": 803391057,
23                                    "ciudad": {
24                                        "id": 2,
25                                        "nombre": "Dongqingou"
26                                    }
27                                }
28                            }
29                        ]
30                    }
31                }
32            ]
33        }
34    }
35 }
```

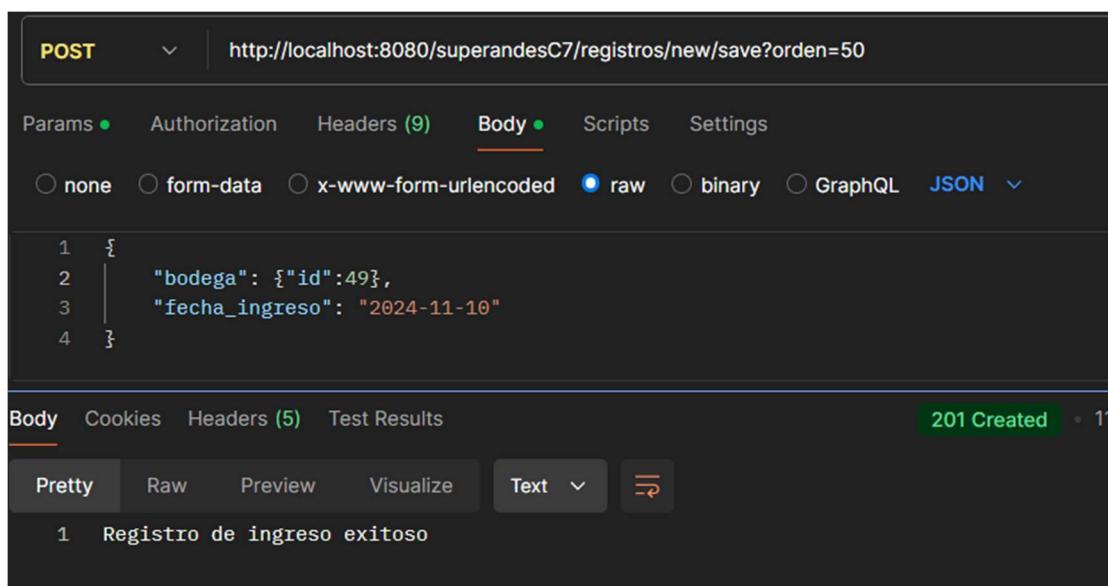
## NUEVO RF10 - Registrar ingreso de productos a la bodega

```
/**RF 10 */
@Modifying
@Query(value = "INSERT INTO registros (orden, fecha_ingreso, bodega) VALUES ( :orden , :fecha_ingreso, :bodega)", nativeQuery = true)
void insertarRegistro(@Param("orden") Long orden, @Param("fecha_ingreso") Date fecha_ingreso,
@Param("bodega") Long bodega);
```

Este requerimiento permite registrar el ingreso de productos a la bodega en la base de datos. El método insertarRegistro ejecuta una consulta SQL en la cual se ingresa orden (id de la orden de compra), fecha de ingreso (Date) y bodega (id de la bodega).

Para verificar que el Api permite registrar el ingreso de productos a la bodega en la base de datos correctamente, se utiliza una solicitud de tipo POST dirigida a la URL:

<http://localhost:8080/superandesC7/registros/new/save?orden={orden}>



```
POST http://localhost:8080/superandesC7/registros/new/save?orden=50
```

Body

```
1 {
2     "bodega": {"id":49},
3     "fecha_ingreso": "2024-11-10"
4 }
```

Body

201 Created

```
1 Registro de ingreso exitoso
```

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

El llamado a este path también hace una verificación de la orden y bodega seleccionada. En caso de que la bodega no pertenezca a la sucursal de la orden, o si la fecha es anterior a la de creación de la orden, esta no prosigue dada la inconsistencia.

HTTP Proyecto / Registros / Crear registro 2 Save

POST <http://localhost:8080/superandesC7/registros/new/save?orden=51>

Params • Authorization Headers (9) Body • Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "bodega": {"id":2},
3   "fecha_ingreso": "2024-11-11"
4 }
```

Body Cookies Headers (4) Test Results 400 Bad Request • 163 ms • 203 B

Pretty Raw Preview Visualize Text

```
1 La bodega no pertenece a la sucursal de destino de la orden
```

Además, al momento de actualizar los precios promedios de cada producto modificando o creando su tupla en la tabla de Almacenajes, la función llamará a un rollback si se intenta ingresar a la bodega más unidades del producto x de las que puede soportar, avisándole en el error al usuario la bodega y producto en cuestión.

POST <http://localhost:8080/superandesC7/registros/new/save?orden=51> Send

Params • Authorization Headers (9) Body • Scripts Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "bodega": {"id":49},
3   "fecha_ingreso": "2024-11-11"
4 }
```

Body Cookies Headers (4) Test Results 500 Internal Server Error • 13.31 s • 6.92 KB •

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2024-11-04T00:53:21.392+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "trace": "java.sql.SQLException: La cantidad de productos 6 a ingresar supera la capacidad de la bodega
49\r\n\tat uniandes.edu.co.proyecto.controller.RegistroController.registroGuardar"
```

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

RFC 1 - Mostrar el índice de ocupación de cada una de bodegas de una sucursal

```
/*RFC1 */
public interface RespuestaIndiceOcupacion{
    Long getBodega();
    Long getProducto();
    int getIndice_ocupacion();
}
```

```
@Query(value = "SELECT almacenajes.bodega, almacenajes.producto, productos.volumen*almacenajes.cantidad/almacenajes.capacidad indice_ocupacion
Collection<RespuestaIndiceOcupacion> darIndiceOcupacion(@Param(\"sucursal\") Long sucursal);"
```

```
FROM almacenajes INNER JOIN productos ON almacenajes.producto = productos.cod_barras INNER JOIN bodegas on almacenajes.bodega = bodegas.id WHERE bodegas.sucursal = :sucursal",
```

Este requerimiento permite obtener el índice de ocupación de las bodegas de una sucursal en la base de datos. El método darIndiceOcupación ejecuta una consulta SQL en la cual se muestra la bodega, el producto, y la división entre el volumen del producto por su cantidad y la capacidad (todos de la tabla almacenajes, menos la bodega). Se hace a través de la tabla almacenajes, inner join con la tabla productos, uniéndoles por el PK del producto, y un inner join con la tabla bodegas, uniéndoles por el PK de la bodega. Cuenta con un parámetro que es la id de la sucursal, de la cual se mostrarán los índices de ocupación.

Para verificar que el API permita obtener el índice de ocupación de las bodegas de una sucursal en la base de datos correctamente, se utiliza una solicitud de tipo GET dirigida a la URL:

<http://localhost:8080/superandesC7/bodegas/indiceOcupacionPorProducto/{sucursal}> , en este caso la 1

The screenshot shows a Postman request configuration for a GET request to the URL `http://localhost:8080/superandesC7/bodegas/indiceOcupacionPorProducto/1`. The request includes parameters, headers, and a body. The response status is 200 OK, and the response body is displayed in JSON format, showing a list of three warehouse occupancy indices.

indice	producto	bodega	indice_ocupacion
1	2	7	6248
2	3	7	9077
3	8	1	2455
4	2	1	3045

Nicolás Ballén-202310273  
María Juliana Ballesteros-202313216  
Mauricio Martinez-202314461

RFC 2 - Mostrar los productos que cumplen con cierta característica

```
/*RFC2.1 Precio en rango*/  
@Query(value = "SELECT productos.* FROM productos WHERE precio_unitario BETWEEN :menor AND :mayor", nativeQuery=true)  
Collection<Producto> darProductoRangoPrecios(@Param("menor") int menor, @Param("mayor") int mayor);  
  
/*RFC2.2 Vencimiento posterior a*/  
@Query(value = "SELECT productos.* FROM productos WHERE fecha_vencimiento > TO_DATE(:fecha, 'YYYY-MM-DD')", nativeQuery=true)  
Collection<Producto> darProductoPosterior(@Param("fecha") String fecha);  
  
/*RFC2.3 Vencimiento anterior a*/  
@Query(value = "SELECT productos.* FROM productos WHERE fecha_vencimiento < TO_DATE(:fecha, 'YYYY-MM-DD')", nativeQuery=true)  
Collection<Producto> darProductoAnterior(@Param("fecha") String fecha);  
  
/*RFC2.4 Disponible en x sucursal*/  
@Query(value = "SELECT productos.* FROM productos INNER JOIN almacenes ON productos.cod_barra=almacenes.producto INNER JOIN bodegas ON bodegas.id=almacenes.bodega  
Collection<Producto> darProductoSucursal(@Param("sucursal") Long sucursal);  
  
WHERE bodegas.sucursal = :sucursal", nativeQuery=true)  
  
/*RFC2.5 Categoria*/  
@Query(value = "SELECT productos.* FROM productos INNER JOIN categorias ON productos.categoria=categorias.codigo WHERE categorias.codigo = :categoria", nativeQuery=true)  
Collection<Producto> darProductoCategoria(@Param("categoria") Long categoria);
```

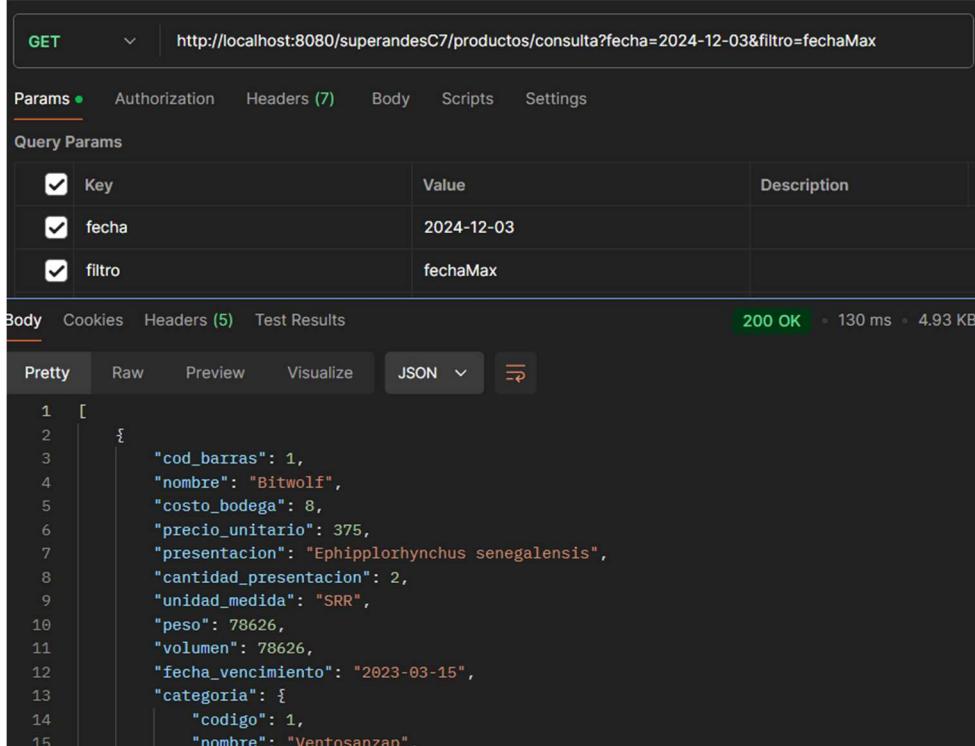
Este requerimiento permite obtener los productos que cumplen con ciertas características en la base de datos. Este requerimiento permite al usuario ver una lista de productos que depende de una condición también impuesta por el usuario. Para el de rango de precio, está darProductoRangoPrecios, el cual muestra la información de los productos de la tabla productos, donde el rango de precio está entre 2 valores ingresados por el usuario. Para fecha de vencimiento posterior, está darProductoPosterior, el cual muestra la información de los productos de la tabla productos, donde la fecha de vencimiento es mayor a la ingresada por el usuario. Para fecha de vencimiento anterior, está darProductoAnterior, el cual muestra la información de los productos de la tabla productos, donde la fecha de vencimiento es menor a la ingresada por el usuario. Para los que están disponibles en una sucursal, está darProductoSucursal, el cual muestra la información de los productos de la tabla productos, donde la sucursal sea igual a la ingresada por el usuario. Para los que son de una categoría específica, está darProductoCategoria, el cual muestra la información de los productos de la tabla productos, donde la categoría sea igual a la ingresada por el usuario.

Para verificar que el Api permita obtener los productos que cumplen con ciertas características en la base de datos correctamente, se utiliza una solicitud de tipo GET dirigida a la URL:

<http://localhost:8080/superandesC7/productos/consulta?menor={menor}&mayor={mayor}>  
[http://localhost:8080/superandesC7/productos/consulta?fecha={"fecha}](http://localhost:8080/superandesC7/productos/consulta?fecha={)  
<http://localhost:8080/superandesC7/productos/consulta?Sucursal={sucursal}>  
<http://localhost:8080/superandesC7/productos/consulta?categoria={categoria}>

Como parametron adicional esta “filtro”, el cual define por que caracteristica se van a filtrar los productos de las anteriores mencionadas (ignorando los valores de los demás parametros)

Nicolás Ballén-202310273  
 María Juliana Ballesteros-202313216  
 Mauricio Martínez-202314461



Key	Value	Description
fecha	2024-12-03	
filtro	fechaMax	

```

1 [ 
2   {
3     "cod_barras": 1,
4     "nombre": "Bitwolf",
5     "costo_bodega": 8,
6     "precio_unitario": 375,
7     "presentacion": "Ephippiorhynchus senegalensis",
8     "cantidad_presentacion": 2,
9     "unidad_medida": "SRR",
10    "peso": 78626,
11    "volumen": 78626,
12    "fecha_vencimiento": "2023-03-15",
13    "categoria": {
14      "codigo": 1,
15      "nombre": "Ventosanzap",
16    }
17  ]
  
```

### RFC 3 - Inventario de productos en una bodega

```

/*RFC3 */
public interface RespuestaInventario{
    Long getProducto();
    Long getCapacidad();
    Long getCantidad();
    Long getN_minimo();
    Long getCosto();
}

@Query(value = "SELECT almacenajes.producto producto, almacenajes.capacidad capacidad, almacenajes.cantidad cantidad, almacenajes.costo_promedio costo,
Collection<RespuestaInventario> darInventarioBodega(@Param(\"bodega\") Long bodega);"

almacenajes.nivel_minimo n_minimo FROM almacenajes WHERE bodega = :bodega", nativeQuery = true)
  
```

Este requerimiento permite obtener el inventario de productos de una bodega en la base de datos. El método darInventario ejecuta una consulta SQL en la cual se muestra el producto, capacidad, cantidad y costo promedio a partir de la tabla de Almacenajes. Cuenta con un parámetro que es la id de la bodega, de la cual se mostrarán los productos.

Para verificar que el Api permita obtener el inventario de productos de una bodega en la base de datos correctamente, se utiliza una solicitud de tipo GET dirigida a la URL:

<http://localhost:8080/superandesC7/bodegas/{bodega}/inventario>

Con ello mostramos los datos básicos de la bodega, así como el inventario compuesto de los detalles de cada producto dentro de la bodega, es decir, su registro de almacenaje limitado como se ve en la interfaz RespuestaInventario.

Nicolás Ballén-202310273  
María Juliana Ballesteros-202313216  
Mauricio Martínez-202314461

The screenshot shows a Postman interface with a GET request to `http://localhost:8080/superandesC7/bodegas/1/inventario`. The response status is 200 OK. The response body is a JSON object with the following structure:

```
1  {
2      "inventario": [
3          {
4              "producto": 2,
5              "cantidad": 3707,
6              "capacidad": 95693,
7              "n_minimo": 6613,
8              "costo": 78850
9          },
10         {
11             "producto": 8,
12             "cantidad": 1840,
13             "capacidad": 58922,
14             "n_minimo": 1467,
15             "costo": 2076
16         }
17     ],
18     "bodega": {
19         "id": 1,
20         "nombre": "Little Purple Monkeyflower",
21         "tamaño": 3805,
22         "sucursal": {
23             "Postbot"
24         }
25     }
26 }
```

RFC 4 - Mostrar las sucursales en las que hay disponibilidad de un producto

```
/*RFC4 */
@Query(value = "SELECT DISTINCT sucursales.* FROM sucursales INNER JOIN bodegas ON bodegas.sucursal=sucursales.id INNER JOIN almacenajes ON bodegas.id=almacenajes.bodega", nativeQuery = true)
Collection<Sucursal> darSucursalesConProducto(@Param("producto") long producto);
```

```
WHERE almacenajes.producto = :producto", nativeQuery = true)
```

Este requerimiento permite obtener las sucursales en las que hay disponibilidad de un producto en la base de datos. El método `darSucursalesConProducto` ejecuta una consulta SQL en la cual se muestra la información de las sucursales. Se hace a través de un inner join con la tabla `almacenajes` y `bodegas`, uniéndoles por el PK de las bodegas pertenecientes a la sucursal. El parámetro es el producto, del cual el usuario quiere saber las sucursales con bodegas que lo tengan (registros de almacenaje de dicho producto para dichas bodegas).

Para verificar que el API permita obtener las sucursales en las que hay disponibilidad de un producto la base de datos correctamente, se utiliza una solicitud de tipo GET dirigida a la URL:

<http://localhost:8080/superandesC7/sucursales/consulta?producto={producto}>

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

The screenshot shows a Postman interface with a GET request to `http://localhost:8080/superandesC7/sucursales/consulta?producto=1`. The response body is a JSON array with two elements:

```
[{"id": 1, "nombre": "Saribudolok", "ciudad": {"id": 1, "nombre": "Lien Junction"}, "telefono": 721465833, "direccion": "6 Lien Junction", "tamaño": 19}, {"id": 8, "nombre": "Shoshone", "ciudad": {"id": 9, "nombre": "Semikarakorsk"}, "telefono": 156879693, "direccion": "11476 Bluestem Drive", "tamaño": 19549}]
```

RFC 5 - Mostrar todos los productos que requieren una orden de compra

```
public interface RespuestaInsuficiente{
    Long getId();
    String getNombre();
    Long.getBodega();
    Long.getProveedor();
    Long.getSucursal();
    int getCantidad();
}
@Query(value = "SELECT DISTINCT productos.cod_barras id, productos.nombre nombre, almacenajes.bodega bodega, ofertas.proveedor proveedor, bodegas.sucursal sucursal,
Collection<RespuestaInsuficiente> darProductoInsuficiente();"

almacenajes.cantidad cantidad FROM productos INNER JOIN almacenajes ON productos.cod_barras=almacenajes.producto INNER JOIN bodegas ON bodegas.id = almacenajes.bodega
INNER JOIN ofertas ON productos.cod_barras=ofertas.producto WHERE almacenajes.cantidad<almacenajes.nivel_minimo", nativeQuery=true)
```

Este requerimiento permite obtener todos los productos que requieren una orden de compra en la base de datos. El método `darProductoInsuficiente` ejecuta una consulta SQL en la cual se muestra el código de barras, nombre, bodega, proveedor, sucursal, cantidad. Se hace a través de un inner join de la tabla `productos` con la tabla `almacenajes`, uniéndoles por el PK del producto, un inner join con `bodegas`, uniéndoles por el PK de `bodega`, y un inner join con `ofertas`, uniéndoles por el PK del producto, donde la cantidad sea menor al nivel mínimo. Comparando la cantidad del producto en el almacenaje de cada bodega que lo tiene con el nivel mínimo que dicha bodega tiene para el mismo, se obtienen aquellos que no satisfacen el nivel mínimo en cada bodega junto a un proveedor existente según las ofertas.

Para verificar que el API permita obtener todos los productos que requieren una orden de compra en la base de datos correctamente, se utiliza una solicitud de tipo GET dirigida a la URL:

<http://localhost:8080/superandesC7/productos/insuficientes>

Nicolás Ballén-202310273  
 María Juliana Ballesteros-202313216  
 Mauricio Martinez-202314461

```

GET      http://localhost:8080/superandesC7/productos/insuficientes
Params   Authorization   Headers (7)   Body   Scripts   Settings
Body   Cookies   Headers (5)   Test Results   200 OK
Pretty  Raw   Preview   Visualize   JSON   ↻
57     },
58     {
59       "id": 4,
60       "nombre": "Zoolab",
61       "sucursal": 8,
62       "proveedor": 9,
63       "cantidad": 2417,
64       "bodega": 9
65     },
66     {
67       "id": 4,
68       "nombre": "Zoolab",
69       "sucursal": 8,
70       "proveedor": 6,
71       "cantidad": 2417,
72       "bodega": 9
73     },
  
```

### NUEVO RFC6 – Consulta de documentos de ingreso de productos a bodega – SERIALIZABLE

El método registrosMesSR realiza una consulta para obtener los registros de ingresos a una bodega en un mes específico, junto con información del proveedor y de la orden asociada. La operación se ejecuta en una transacción con un nivel de aislamiento SERIALIZABLE para evitar conflictos de concurrencia y hacer rollback en caso de una SQLException. La consulta SQL selecciona fecha\_ingreso, proveedor y orden de las tablas registros y ordenes, uniendo ambas tablas en el campo orden. Además, filtra los resultados para fechas mayores o iguales a la fecha proporcionada en el parámetro (definida luego en el controlador como aquellos 30 días antes de la fecha actual), utilizando el formato YYYY-MM-DD, y restringe la búsqueda a una bodega específica según el parámetro bodega (perteneciente a la sucursal luego definida en el path del controller). El método devuelve una colección de objetos RespuestaConsultaMes que contienen los datos solicitados.

```

/*RFC 6 */
@Transactional(isolation = Isolation.SERIALIZABLE, rollbackFor = SQLException.class)
@Query(value = "SELECT registros.fecha_ingreso fecha, ordenes.proveedor proveedor, registros.orden orden
FROM registros inner join ordenes on ordenes.id = registros.orden where fecha_ingreso >= TO_DATE(:fecha, 'YYYY-MM-DD') AND bodega = :bodega", nativeQuery = true)
Collection<RespuestaConsultaMes> registrosMesSR(@Param("fecha") String fecha, @Param("bodega") Long bodega);
  
```

Nicolás Ballén-202310273

María Juliana Ballesteros-202313216

Mauricio Martínez-202314461

```
@GetMapping("/registros/committed/{sucursal}/{bodega}")
@Transactional(isolation = Isolation.READ_COMMITTED, rollbackFor = SQLException.class)
public ResponseEntity<Object> registros30Committed(@PathVariable("bodega") Long bodega, @PathVariable("sucursal") Long sucursal) throws SQLException {
    try {
        Bodega bodegaElegida = bodegaRepository.darBodega(bodega);
        if (bodegaElegida == null) {
            throw new SQLException("La bodega no existe");
        }
        else if (bodegaElegida.getSucursal().getId() != sucursal) {
            throw new SQLException("La bodega no pertenece a la sucursal");
        }
        Calendar fecha = Calendar.getInstance();
        fecha.add(Calendar.DAY_OF_MONTH, -30);
        DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
        String fechaInput = df.format(fecha.getTime());
        Thread.sleep(30 * 1000);
        Collection<RespuestaConsultaMes> registros = registroRepository.registrosMesRC(fechaInput, bodega);
        Map<String, Object> response = new HashMap<>();
        response.put("registros", registros);
        response.put("bodega", bodegaElegida.getNombre());
        response.put("sucursal", bodegaElegida.getSucursal().getNombre());
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        throw new SQLException("Error en la lectura de los registros");
    }
}
```

Véase como la bodega debe pertenecer a la sucursal escogido, pues de lo contrario no se prosigue con la consulta. En el caso positivo, se impondrá un temporizador de 30 segundos antes del llamado de la sentencia SQL, permitiendo así las pruebas de concurrencia que explicaremos más adelante.

## NUEVO RFC7 – Consulta de documentos de ingreso de productos a bodega – READ\_COMMITTED

Este obtiene exactamente el mismo resultado que RFC6, con la diferencia de tener definido un nivel de aislamiento diferente 8READ\_COMMITTED):

```
/**RFC 7 */
@Transactional(isolation = Isolation.READ_COMMITTED, rollbackFor = SQLException.class)
@Query(value = "SELECT registros.fecha_ingreso fecha, ordenes.proveedor, registros.orden orden FROM registros inner join ord
Collection<RespuestaConsultaMes> registrosMesRC(@Param("fecha") String fecha, @Param("bodega") Long bodega);
```

GET http://localhost:8080/superandesC7/registros/serial/1/49

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2      "sucursal": "Onsgard",
3      "registros": [
4          {
5              "proveedor": 1,
6              "orden": 50,
7              "fecha": "2024-11-10T00:00:00.000+00:00"
8          },
9      ],
10     "bodega": "BodegaRegistroPrueba"
11 }
```

Nicolás Ballén-202310273  
 María Juliana Ballesteros-202313216  
 Mauricio Martínez-202314461

## Escenarios de Prueba de Concurrencias:

### Escenario de prueba de concurrencia 1

Sesión 1 - RFC6	Tiempo	Sesión 2 - RF10
Se ejecuta RFC6	t1	
	t2 (Antes de que RFC6 termine de ejecutarse)	Se ejecuta RF10
RFC6 se sigue ejecutando	t3	RF10 en espera
RFC6 termina de ejecutarse	t4	
	t5	RF10 ingresa el registro

Dado que RFC6 se está ejecutando con un nivel de aislamiento SERIALIZABLE, la transacción que se ejecuta en este requerimiento mantiene un bloqueo sobre los registros que está consultando. Esto quiere decir que el RF10, al intentar insertar un nuevo registro, tiene que esperar a que la ejecución de RFC6 se complete. Esto debido a que, al ser un nuevo ingreso (suponiendo una fecha reciente), se entiende también se ingreso recientemente, entrando dentro de las condiciones del predicado de estar en los últimos 30 días (predicado a partir del cual RF6 establecería un candado de lectura de larga duración).

El resultado es que RF10 no puede insertar el registro hasta que la consulta RFC6 finalice. Por esto mismo **la lectura de RF6 no mostrará el registro ingresado por RF10**, siendo que ocurrirá después de concretarse la lectura.

### Escenario de prueba de concurrencia 2

Sesión 1 - RFC6	Tiempo	Sesión 2 - RF10
Se ejecuta RFC6	t1	
	t2 (Antes de que RFC6 termine de ejecutarse)	Se ejecuta RF10
RFC6 se sigue ejecutando	t3	RF10 inserta un nuevo registro
RFC6 termina de ejecutarse	t4	RF10 terminado
RFC6 muestra los registros, incluido el ingresado por RF10	t5	RF10 terminado

Dado que RFC7 se está ejecutando con un nivel de aislamiento READ\_COMMITED, la transacción que se ejecuta en este requerimiento no impone candados de lectura de larga sino de corta duración. Esto quiere decir que RF10 no tiene candado alguno que le impida ingresar sus registros, siendo que hasta ese punto todo candado de lectura de RFC7 ya se habría soltado al instante.

Con esto en mente, el resultado es que RF10 si puede insertarse el registro antes de que la consulta RFC7 finalice. Por esto mismo, **la lectura de RF6 si mostrará el registro ingresado por RF10**, siendo que ocurrirá antes de concretarse la lectura.

Nicolás Ballén-202310273  
María Juliana Ballesteros-202313216  
Mauricio Martinez-202314461

## Resultado de Pruebas de Concurrencias:

Con tal de mantener mejor control sobre el ingreso de los registros, cada prueba se realizó sobre una nueva orden de compra dirigida una nueva bodega creada específicamente para la sucursal destino de la primera. Por la misma razón, toda consulta de registros previa a RF10 será vacía, en tanto hablamos de una bodega nueva creada específicamente para la prueba. De ahí que no hagamos revisiones o lecturas previas.

### Escenario de prueba de concurrencia 1-RFC6 +RF10:

Iniciando con la lectura RF6 no tuvimos problemas, mostrando el delay del temporizador como se esperaba:

GET http://localhost:8080/superandesC7/registros/serial/1/49

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK 1 m 31.56 s

Pretty Raw Preview Visualize JSON

```
1 {  
2   "sucursal": "Onsgard",  
3   "registros": [  
4     {  
5       "proveedor": 1,  
6       "orden": 50,  
7       "fecha": "2024-11-10T00:00:00.000+00:00"  
8     }  
9   ],  
10  "bodega": "BodegaRegistroPrueba"
```

Sending request...

Sin embargo, al momento de intentar el registro no denotamos bloqueo alguno. Si bien se demoró en su momento un poco más de lo usual, el registro de ingreso paso de todos modos antes de terminar el contador:

```
1 {  
2   "bodega": {"id":49},  
3   "fecha_ingreso": "2024-11-10"  
4 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

```
1 Registro de ingreso exitoso
```

Precisamente, y como nos temíamos, el registro se mostro en la lectura a pesar del nivel de aislamiento:

Nicolás Ballén-202310273  
María Juliana Ballesteros-202313216  
Mauricio Martínez-202314461

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ↻

```
1  {
2      "sucursal": "Onsgard",
3      "registros": [
4          {
5              "orden": 50,
6              "proveedor": 1,
7              "fecha": "2024-11-10T00:00:00.000+00:00"
8          },
9      ],
10     "bodega": "BodegaRegistroPrueba"
11 }
```

Desconocemos la razón de la falla del nivel de aislamiento, siendo que como se ven en las capturas este fue serializable y el temporizador se ubicó antes del llamado a la sentencia SQL. En todo caso, reconocemos la necesidad de mirar más a fondo las posibles razones del error, bien sean en la definición del nivel de aislamiento, o en el uso de la función `thread.sleep()`.

#### **Escenario de prueba de concurrencia 1-RFC7 +RF10:**

En este caso, dado posiblemente al nivel de aislamiento predeterminado de Oracle siendo precisamente `READ_COMMITTED`, si obtuvimos los resultados deseados en el segundo escenario. Si bien se obtuvo el mismo resultado que en el primer escenario, este habría sido el esperado con este nivel de aislamiento, permitiendo insertar el registro de RF10 y mostrándolo como corresponde en la lectura de RF7. Cabe mencionar que en repeticiones del escenario algunos ingresos se sintieron más rápido que con el escenario 1, por lo cual de pronto si hubo efecto alguno en los niveles de aislamiento anteriores, siendo un posible manejo erróneo de `thread.sleep()` lo que evitó el correcto funcionamiento de estos.