202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

ENTREGA PROYECTO 2

SISTEMAS TRANSACCIONALES

2024-10

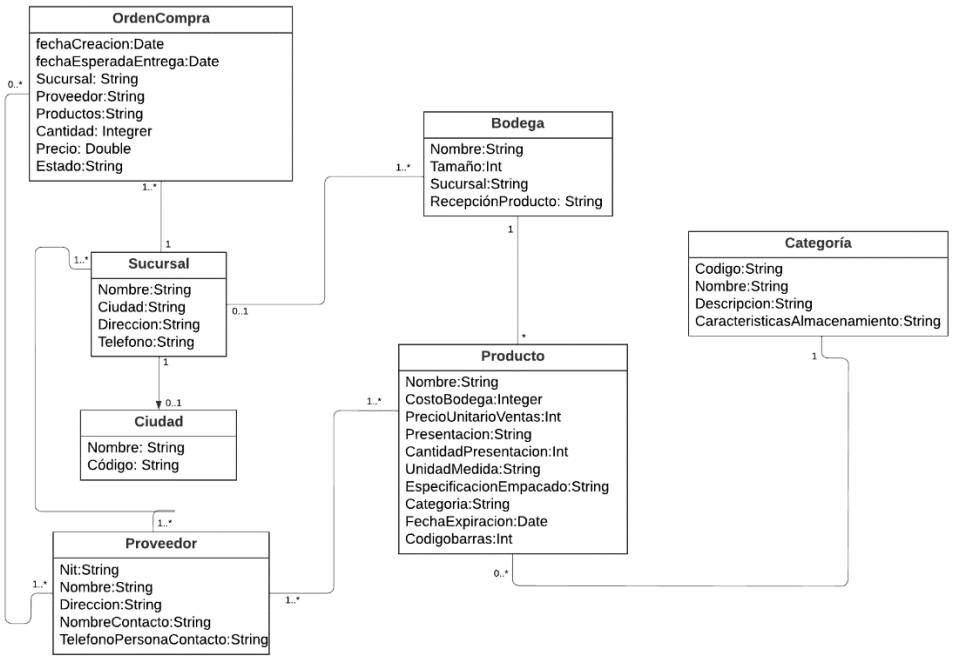
UNIVERSIDAD DE LOS ANDES

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

TABLA DE CONTENIDOS

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

Análisis y revisión del modelo de datos: modelos UML y relacional y su descripción, indicando las modificaciones hechas.



Modificaciones realizadas:

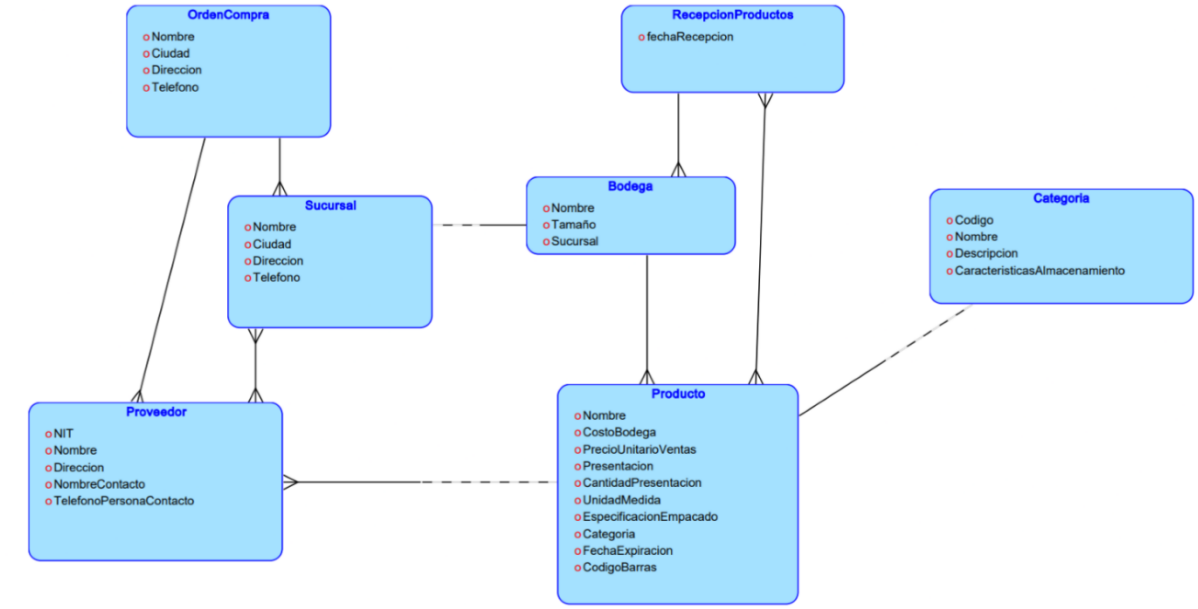
UML:

Las clases actualizadas en el modelo son Ciudad y OrdenDeCompra. Agregamos la clase Ciudad, como fue sugerido, y agregamos los atributos faltantes en OrdenDeCompra para su correcto funcionamiento. En cuanto a las relaciones, hicimos los siguientes cambios:

- Una **bodega** tiene de 0 a 1 **sucursal**, lo que significa que una bodega puede estar o no asociada a una sucursal.
- Un **proveedor** debe tener de 1 a muchos **productos**, asegurando que un proveedor siempre esté vinculado a al menos un producto.
- Un **producto** tiene 1 **categoría**, y una **categoría** tiene de 0 a muchos **productos**, lo que significa que cada producto pertenece a una categoría, pero una categoría puede tener múltiples productos.

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

- Un **proveedor** tiene de 0 a muchas **órdenes de compra**, lo que indica que un proveedor puede tener varias órdenes de compra o ninguna.
- Una **bodega** tiene varios **productos**, y un **producto** tiene una única **bodega**, lo que asegura que cada producto esté almacenado en una sola bodega, pero una bodega puede almacenar varios productos.
- Una **orden de compra** tiene una **sucursal**, y una **sucursal** tiene varias **órdenes de compra**, lo que significa que cada orden de compra está asociada a una sucursal, y una sucursal puede gestionar múltiples órdenes de compra.



RELACIONAL (Adjunto en el repositorio)

- La relación entre bodega y sucursal se arregló siendo así que Sucursal debe ser la que tenga la FK, NN de bodega.
- Se elimino toda tabla adicional que no fuera de muchos a muchos.
- La FK de id_producto en la recepción de productos se agregó bajo el comentario anteriormente dado.
- En producto se le agrego una FK de id_bodega para modelar la relación.
- Se cambio el NN de fecha de expiración por DATE.
- Se agrego la justificación sobre UML a relacional.
- Se modifico la normalización para especificar las únicas tablas que se tienen actualmente.

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

Diseño de la base de datos: archivos .sql con los scripts utilizados para la creación del a BD.

```sql
-- Tabla CIUDAD
CREATE TABLE CIUDAD (
    ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    NOMBRE VARCHAR2(100) NOT NULL,
    CODIGO VARCHAR2(10) NOT NULL
);

-- Tabla SUCURSAL (Asociada a una CIUDAD)
CREATE TABLE SUCURSAL (
    ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    NOMBRE VARCHAR2(100) NOT NULL,
    CIUDAD_ID NUMBER NOT NULL,
    DIRECCION VARCHAR2(255),
    TELEFONO VARCHAR2(20),
    CONSTRAINT FK_CIUDAD FOREIGN KEY (CIUDAD_ID) REFERENCES CIUDAD(ID)
);

-- Tabla BODEGA (Asociada a una SUCURSAL)
CREATE TABLE BODEGA (
    ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    NOMBRE VARCHAR2(100) NOT NULL,
    TAMANO NUMBER NOT NULL,  -- Capacidad de la bodega en volumen
    SUCURSAL_ID NUMBER NOT NULL,
    CONSTRAINT FK_SUCURSAL_BODEGA FOREIGN KEY (SUCURSAL_ID) REFERENCES
SUCURSAL(ID)
);

-- Tabla CATEGORIA_PRODUCTO
CREATE TABLE CATEGORIA_PRODUCTO (
    ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    NOMBRE VARCHAR2(100) NOT NULL,
    DESCRIPCION VARCHAR2(255),
    CARACTERISTICAS_ALMACENAMIENTO VARCHAR2(255)
);

-- Tabla PRODUCTO (Asociado a una categoría)
CREATE TABLE PRODUCTO (
    ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    NOMBRE VARCHAR2(100) NOT NULL,
    COSTO_BODEGA NUMBER NOT NULL,
```

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

```
    PRECIO_UNITARIO NUMBER NOT NULL,
    PRESENTACION VARCHAR2(50) NOT NULL,
    CANTIDAD_PRESENTACION NUMBER NOT NULL,
    UNIDAD_MEDIDA VARCHAR2(50) NOT NULL,
    VOLUMEN_EMPAQUE NUMBER,
    PESO_EMPAQUE NUMBER,
    FECHA_EXPIRACION DATE,
    CODIGO_BARRAS VARCHAR2(50) UNIQUE,
    CATEGORIA_ID NUMBER,
    CONSTRAINT FK_CATEGORIA FOREIGN KEY (CATEGORIA_ID) REFERENCES
CATEGORIA_PRODUCTO(ID)
);

-- Tabla PROVEEDOR
CREATE TABLE PROVEEDOR (
    ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    NIT VARCHAR2(50) NOT NULL UNIQUE,
    NOMBRE VARCHAR2(100) NOT NULL,
    DIRECCION VARCHAR2(255),
    NOMBRE_CONTACTO VARCHAR2(100),
    TELEFONO_CONTACTO VARCHAR2(20)
);

-- Tabla ORDEN_DE_COMPRA (Encabezado)
CREATE TABLE ORDEN_DE_COMPRA (
    ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    FECHA_CREACION DATE DEFAULT SYSDATE NOT NULL,
    FECHA_ESPERADA_ENTREGA DATE NOT NULL,
    ESTADO VARCHAR2(50) DEFAULT 'vigente' CHECK (ESTADO IN ('vigente',
'anulada', 'entregada')),
    SUCURSAL_ID NUMBER NOT NULL,
    PROVEEDOR_ID NUMBER NOT NULL,
    CONSTRAINT FK_SUCURSAL_ORDEN FOREIGN KEY (SUCURSAL_ID) REFERENCES
SUCURSAL(ID),
    CONSTRAINT FK_PROVEEDOR_ORDEN FOREIGN KEY (PROVEEDOR_ID) REFERENCES
PROVEEDOR(ID)
);

-- Tabla ORDEN_DE_COMPRA_DETALLE (Detalle de la orden)
CREATE TABLE ORDEN_DE_COMPRA_DETALLE (
    ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    ORDEN_ID NUMBER NOT NULL,
    PRODUCTO_ID NUMBER NOT NULL,
    PRECIO_PRODUCTO NUMBER NOT NULL,
```

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

```
    CANTIDAD_PRODUCTO NUMBER NOT NULL,
    CONSTRAINT FK_ORDEN FOREIGN KEY (ORDEN_ID) REFERENCES ORDEN_DE_COMPRA(ID),
    CONSTRAINT FK_PRODUCTO FOREIGN KEY (PRODUCTO_ID) REFERENCES PRODUCTO(ID)
);

-- Tabla PRODUCTO_BODEGA (Inventario de productos en bodegas específicas)
CREATE TABLE PRODUCTO_BODEGA (
    ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    BODEGA_ID NUMBER NOT NULL,
    PRODUCTO_ID NUMBER NOT NULL,
    CANTIDAD_ACTUAL NUMBER NOT NULL,
    CANTIDAD_MINIMA NUMBER NOT NULL,
    CONSTRAINT FK_BODEGA FOREIGN KEY (BODEGA_ID) REFERENCES BODEGA(ID),
    CONSTRAINT FK_PRODUCTO_BODEGA FOREIGN KEY (PRODUCTO_ID) REFERENCES
PRODUCTO(ID)
);
```

Documentación de las clases de Java Spring y de la arquitectura de la aplicación

Arquitectura de la Aplicación

La arquitectura de esta aplicación Java Spring se basa en una estructura de capas que sigue principios de diseño modular. Cada capa tiene una responsabilidad específica, promoviendo la separación de preocupaciones y facilitando el mantenimiento y escalabilidad del código. Las capas principales son:

1. **Capa de Entidades**: Define las estructuras de datos de la aplicación mediante las entidades JPA. Estas entidades representan las tablas de la base de datos y están configuradas con anotaciones de persistencia que facilitan la relación entre ellas.
2. **Capa de Repositorios**: Proporciona una abstracción para el acceso a los datos. Cada repositorio extiende `JpaRepository`, lo que permite realizar operaciones CRUD sobre las entidades sin necesidad de implementar manualmente consultas SQL.
3. **Capa de Servicios**: Contiene la lógica de negocio de la aplicación. Los servicios orquestan operaciones que involucran múltiples entidades y coordinan la comunicación entre los repositorios y los controladores.
4. **Capa de Controladores**: Define los puntos de entrada de la API REST. Cada controlador se encarga de recibir solicitudes HTTP, delegar el procesamiento a los servicios correspondientes y devolver respuestas en el formato adecuado.

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

Documentación de Clases

Capa de Entidades

- **BodegaEntity**: Representa una bodega dentro del sistema, que almacena productos. Incluye atributos como `nombre` y `tamaño_m2`, así como relaciones con `SucursalEntity`.
- **OrdenDeCompraEntity**: Representa una orden de compra realizada a un proveedor. Incluye información sobre la fecha de emisión, estado de la orden, sucursal y proveedor asociados.
- **OrdenDetalleEntity**: Define los detalles de una orden de compra, especificando el producto y la cantidad solicitada en cada orden.
- **Producto**: Clase que representa un producto, con atributos como nombre, costo en bodega y precio unitario. También incluye relaciones con la categoría y la bodega en la que se encuentra.
- **SucursalEntity**: Define una sucursal que puede tener múltiples bodegas. Contiene información de identificación como nombre, ciudad, dirección y teléfono.
- **CiudadEntity**: Representa la ciudad en la que se ubica una sucursal o bodega, con atributos como `nombre` y `código_postal`.
- **ProductoBodegaEntity**: Clase de relación que enlaza productos con bodegas, especificando en qué bodega se encuentra cada producto.
- **RegistroIngresoEntity**: Representa un registro de ingreso de productos a la bodega, incluyendo la fecha y la cantidad de productos ingresados.
- **DetalleIngresoEntity**: Detalle de un registro de ingreso, especificando el producto y la cantidad ingresada en cada operación.

Capa de Repositorios

- **InventarioRepository**: Proporciona métodos para operaciones CRUD en el inventario, así como consultas personalizadas para obtener productos que requieren orden de compra.
- **ProductoRepository**: Repositorio para la entidad `Producto`, que permite realizar búsquedas de productos por características como precio y fecha de vencimiento.

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

- **ProveedorRepository**: Repositorio que gestiona los proveedores en la base de datos, permitiendo operaciones CRUD y consultas específicas de proveedor.
- **RegistroIngresoRepository**: Repositorio para gestionar el registro de ingresos de productos en la base de datos.
- **BodegaRepository**: Proporciona acceso a los datos de las bodegas.
- **CiudadRepository**: Gestiona la persistencia de las entidades de tipo `CiudadEntity`.
- **SucursalRepository**: Repositorio que administra la entidad `SucursalEntity`, permitiendo el acceso a sus datos.
- **OrdenDeCompraRepository**: Gestiona las órdenes de compra en la base de datos, incluyendo operaciones CRUD y consultas personalizadas.

Capa de Servicios

- **InventarioService**: Gestiona la lógica de negocio del inventario, incluyendo el cálculo de cantidades y costo promedio de productos en bodega.
- **OrdenDeCompraService**: Orquesta la creación, actualización y obtención de órdenes de compra, además de aplicar lógica de negocio específica para el manejo de estas órdenes.
- **ProductoService**: Lógica de negocio para gestionar productos, incluyendo cálculos de precios y control de existencias.
- **RegistroIngresoService**: Lógica de negocio para el registro de ingreso de productos en las bodegas, controlando la cantidad ingresada y actualizando el inventario.
- **BodegaService**: Gestiona las operaciones relacionadas con la bodega, como la creación y actualización de información de bodega.
- **CiudadService**: Lógica de negocio para la gestión de las ciudades en las que se encuentran las sucursales.
- **SucursalService**: Gestiona la información de las sucursales, incluyendo su creación y actualización, además de las relaciones con otras entidades.
- **CascaraService**: Servicio que representa una plantilla genérica para operaciones de negocio adicionales.
- **CategoriaProductoService**: Lógica de negocio para la gestión de categorías de productos, incluyendo su creación y modificación.
- **ProveedorService**: Gestiona la lógica de negocio relacionada con los proveedores, facilitando la creación, actualización y obtención de datos de proveedores.

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

Capa de DTOs (Data Transfer Objects)

- **OrdenDeCompraDTO**: Representa los datos transferidos en las solicitudes y respuestas relacionadas con la orden de compra.
- **OcupacionBodegaDTO**: Contiene datos sobre la ocupación de una bodega, como cantidad de productos y espacio utilizado.
- **BodegaDTO**: Datos de transferencia para la bodega, incluyendo detalles como nombre y tamaño.
- **CiudadDTO**: DTO para la entidad `Ciudad`, usado en las operaciones de solicitud y respuesta que involucran datos de la ciudad.
- **ProductoDetalleDTO**: Contiene detalles sobre un producto específico, como nombre y cantidad en bodega.
- **ProductoDTO**: Representa los datos de un producto que se envían o reciben en la API.
- **SucursalDTO**: DTO para la entidad `Sucursal`, incluye datos como nombre, dirección y teléfono.
- **DetalleIngresoDTO**: Datos de transferencia para el detalle de ingreso de productos, especificando el producto y cantidad ingresada.
- **RegistroIngresoDTO**: Representa los datos de un registro de ingreso de productos a la bodega.
- **InventarioDTO**: Contiene los datos de inventario transferidos en las operaciones API.
- **OrdenDetalleDTO**: Datos de transferencia para los detalles de una orden de compra.

Capa de Controladores

- **BodegaController**: Define los endpoints relacionados con las operaciones de bodega, como creación y consulta de bodegas.
- **InventarioController**: Expone los endpoints para gestionar el inventario, incluyendo operaciones de consulta y actualización.
- **RegistroIngresoController**: Controlador que define los endpoints para registrar ingresos de productos en las bodegas.
- **ProductoController**: Controlador para las operaciones de productos, incluyendo creación, actualización y búsqueda de productos.
- **CiudadController**: Define los endpoints para gestionar las ciudades de las sucursales, incluyendo operaciones CRUD.

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

- **SucursalController**: Controlador que expone los endpoints para operaciones de sucursales, permitiendo la creación y consulta de sucursales.
- **OrdenDeCompraController**: Define los endpoints para gestionar órdenes de compra, permitiendo su creación, modificación y consulta.
- **CategoriaProductoController**: Expone los endpoints para gestionar las categorías de productos.
- **DefaultController**: Controlador general para rutas y operaciones no específicas o rutas por defecto.
- **ProveedorController**: Define los endpoints para gestionar proveedores en el sistema, como creación y actualización de información de proveedores.

Este diseño modular basado en Spring Boot permite una estructura de clases coherente y escalable, facilitando el mantenimiento y la extensibilidad del sistema. Cada clase en su respectiva capa cumple una función específica, permitiendo que el flujo de datos entre la base de datos y los clientes de la API sea eficiente y claro.

ESCENARIO DE PRUEBA DE CONCURRENCIA 1

Objetivo

Evaluar el comportamiento concurrente del sistema al ejecutar RFC6 y RF10 en paralelo, verificando si RF10 debe esperar a que RFC6 complete su operación antes de registrar el ingreso de productos a la bodega.

1. Pasos para la ejecución concurrente de RFC6 y RF10 en la línea de tiempo:

1. **Paso 1 (Usuario A)**: Se ejecuta **RFC6** - La consulta de documentos de ingreso de productos a bodega.
    a. La consulta **RFC6** inicia y establece un aislamiento serializable para evitar inconsistencias en la lectura.
    b. La consulta se ejecutará durante 30 segundos (debido al temporizador `thread.sleep(30000)`), antes de recuperar la información de los documentos.
2. **Paso 2 (Usuario B)**: Antes de que finalicen los 30 segundos de **RFC6**, se ejecuta **RF10** - El registro de ingreso de productos a la bodega.
    a. **RF10** intenta realizar tres operaciones transaccionales:

a) Registrar los detalles del ingreso en la base de datos,
b) Actualizar el inventario y el costo promedio del producto en la bodega,
c) Cambiar el estado de la orden de compra a "ENTREGADA".

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

2. Descripción de lo sucedido:

- Dado que **RFC6** utiliza el nivel de aislamiento **serializable**, se bloquean todas las operaciones de lectura y escritura en las tablas involucradas mientras la consulta está en curso.
- Debido a esto, **RF10** no puede acceder a las tablas relacionadas con el inventario y los documentos de ingreso hasta que **RFC6** termine.
- **RF10** permanece en espera (bloqueado) durante los 30 segundos que tarda **RFC6** en completar su ejecución.

3. Resultado de RFC6:

- Después de que pasan los 30 segundos y **RFC6** finaliza, **RF10** puede proceder con sus operaciones.
- La consulta de **RFC6** muestra los documentos de ingreso de los últimos 30 días para la bodega y la sucursal seleccionadas, sin incluir el nuevo documento de ingreso creado en **RF10**.
- Esto se debe a que **RF10** solo se pudo ejecutar después de que **RFC6** liberó los recursos, por lo que **RFC6** no muestra el documento de ingreso recién registrado.

Conclusión del Escenario de Concurrencia

Este escenario de prueba demostró que, debido al nivel de aislamiento serializable, **RF10** debió esperar a que **RFC6** completara su ejecución para poder registrar el ingreso de productos. Además, **RFC6** no muestra el documento de ingreso realizado por **RF10** en su resultado, ya que este se generó después de finalizar la consulta.

Resultados de RFC6:

La respuesta de RFC6 muestra los documentos de ingreso de los últimos 30 días hasta el momento de su ejecución. No incluye el nuevo registro que RF10 intenta crear, ya que RF10 no se ejecuta hasta después de que RFC6 ha terminado.

Datos Devueltos:

[
 {
  "id": 1001,

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

"fechaIngreso": "2024-10-05",
"sucursal": { "id": 1, "nombre": "Sucursal Norte" },
"bodega": { "id": 2, "nombre": "Bodega Principal" },
"proveedor": { "id": 3, "nombre": "Proveedor A" },
"detalles": [
{
"productoId": 300,
"cantidad": 20,
"precioUnitario": 1200.0
}
]
}
]

Resultados de RF10:

La respuesta de RF10 se genera después de que RFC6 ha finalizado.

RF10 registra un nuevo ingreso en la tabla registro_ingreso, y la respuesta incluye los detalles de este nuevo registro.

Datos Devueltos:

json

Copiar código

{
"id": 1002,
"fechaIngreso": "2024-11-01",
"sucursal": { "id": 1, "nombre": "Sucursal Norte" },
"bodega": { "id": 2, "nombre": "Bodega Principal" },
"proveedor": { "id": 4, "nombre": "Proveedor B" },

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

"detalles": [
{
"productoId": 301,
"cantidad": 15,
"precioUnitario": 1500.0
}
]
}

ESCENARIO DE PRUEBA DE CONCURRENCIA 2:

Objetivo: Evaluar el comportamiento de concurrencia cuando se ejecuta el **RFC7** (consulta de documentos de ingreso de productos a bodega con aislamiento *Read Committed*) y, dentro de los 30 segundos de ejecución de esta consulta, se ejecuta de manera concurrente el **RF10** (registro de ingreso de productos a la bodega en una transacción).

Pasos para la Ejecución Concurrente de RFC7 y RF10 a través de la Línea de Tiempo

1. **Inicio de RFC7**:
    a. El usuario ejecuta la consulta **RFC7** en el sistema.
    b. La consulta **RFC7** inicia con el nivel de aislamiento *Read Committed*.
    c. La transacción asocia el *Id* de la sucursal y el *Id* de la bodega solicitados, devolviendo la lista de documentos de ingreso de los últimos 30 días.
    d. Se activa un temporizador de 30 segundos (usando `Thread.sleep(30000)`) antes de iniciar las sentencias SQL, para simular una larga ejecución y permitir pruebas de concurrencia.
2. **Ejecución Concurrente de RF10**:
    a. Mientras **RFC7** aún está en ejecución (dentro de los 30 segundos de espera), el usuario inicia **RF10**, que registra el ingreso de productos en la bodega correspondiente.
    b. La operación **RF10** sigue una transacción con múltiples pasos:
        i. Registro del ingreso del producto en la bodega.

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

   ii. Actualización del inventario y el costo promedio del producto en la bodega.
   iii. Cambio del estado de la orden de compra a "ENTREGADA".
c. **RF10** realiza un *commit* solo si todas las operaciones se ejecutan correctamente; de lo contrario, hace un *rollback*.

3. **Finalización de RFC7**:
   a. Pasados los 30 segundos de temporizador, **RFC7** continúa y ejecuta la consulta para obtener los documentos de ingreso de productos en la bodega.

Descripción de lo Sucedido

- **Concurrencia y Bloqueo**:
  - Como **RFC7** está ejecutándose con un nivel de aislamiento *Read Committed*, permite las lecturas de datos que ya han sido confirmados (commit), pero también puede leer datos que podrían ser modificados por otras transacciones en curso (por ejemplo, por **RF10**).
  - **RF10** no necesariamente tiene que esperar a que **RFC7** termine para ejecutar el registro de ingreso. Si **RF10** hace un commit antes de que **RFC7** complete su consulta, **RFC7** podrá ver el nuevo ingreso de producto, ya que *Read Committed* permite visualizar datos confirmados por otras transacciones que se completaron mientras **RFC7** estaba en ejecución.
- **Impacto en RFC7**:
  - Debido a que **RFC7** utiliza *Read Committed*, puede ocurrir una lectura no repetible: si **RF10** realiza un commit durante la consulta, **RFC7** puede ver los cambios al momento de leer la base de datos.
  - En este caso, **RFC7** reflejaría los datos actualizados en su resultado final, lo cual incluiría el nuevo documento de ingreso creado por **RF10**.

Resultado Presentado por RFC7

- **Resultados de la Consulta RFC7**:
  - Al finalizar la ejecución de **RFC7**, el resultado incluye:
    - El nombre de la sucursal y el nombre de la bodega solicitados.
    - Un listado de los documentos de ingreso de productos en la bodega en los últimos 30 días.

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

    - En caso de que **RF10** haya completado la transacción mientras **RFC7** estaba en espera, el nuevo documento de ingreso registrado por **RF10** estará presente en el listado que **RFC7** devuelve.
- **Conclusión**:
    - La consulta **RFC7** no bloquea a **RF10**, permitiendo que el ingreso del nuevo producto se realice de forma concurrente.
    - Dado el nivel de aislamiento *Read Committed*, el nuevo documento de ingreso puede aparecer en los resultados de **RFC7** si **RF10** hace un commit antes de que **RFC7** termine su consulta.
    - **RFC7** evita lecturas sucias, pero permite lecturas no repetibles y registros fantasma, lo cual explica por qué el nuevo documento de **RF10** puede aparecer en el resultado de **RFC7** de manera concurrente.

Resultados de RFC7:
La respuesta de RFC7 muestra los documentos de ingreso de los últimos 30 días hasta el momento de su ejecución. No incluye el nuevo ingreso que RF10 intenta crear, ya que RFC7 se ejecuta sin esperar a que RF10 finalice.
Datos Devueltos:


```
[
 {
  "id": 1001,
  "fechaIngreso": "2024-10-05",
  "sucursal": { "id": 1, "nombre": "Sucursal Norte" },
  "bodega": { "id": 2, "nombre": "Bodega Principal" },
  "proveedor": { "id": 3, "nombre": "Proveedor A" },
  "detalles": [
   {
    "productoId": 300,
    "cantidad": 20,
    "precioUnitario": 1200.0
   }
  ]
 }
]
```


Resultados de RF10:
La respuesta de RF10 se genera inmediatamente después de su ejecución.
RF10 registra un nuevo ingreso en la tabla registro_ingreso, y la respuesta incluye los detalles de este nuevo registro.

202221038-Daniel Mancilla
202215891- Jose Salgado
202215816- Angelica Yeraldin Rodríguez

Datos Devueltos:

{
  "id": 1003,
  "fechaIngreso": "2024-11-01",
  "sucursal": { "id": 1, "nombre": "Sucursal Norte" },
  "bodega": { "id": 2, "nombre": "Bodega Principal" },
  "proveedor": { "id": 5, "nombre": "Proveedor C" },
  "detalles": [
    {
      "productoId": 302,
      "cantidad": 10,
      "precioUnitario": 1400.0
    }
  ]
}