Anotaciones sobre modelado

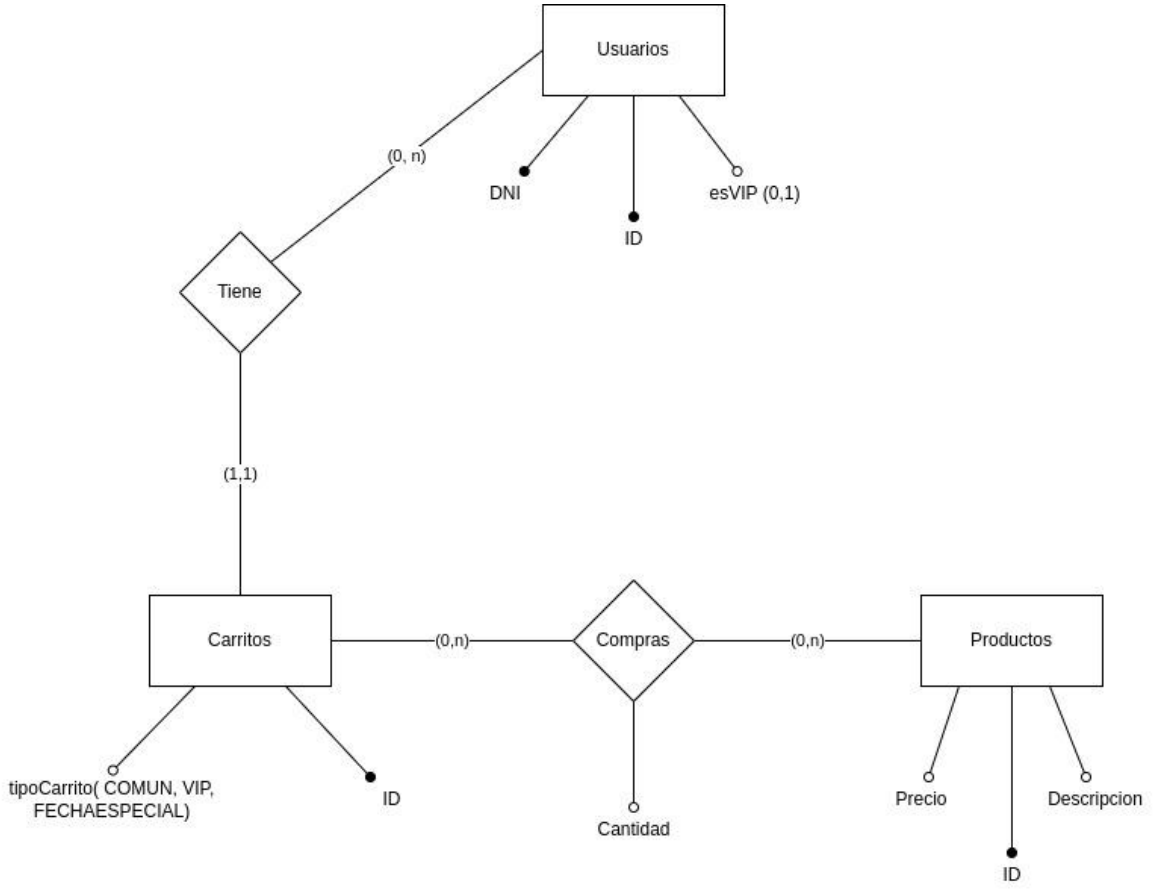
Carrito de compras - Desafio Danaide

Cao Agustin

Sobre el modelado conceptual

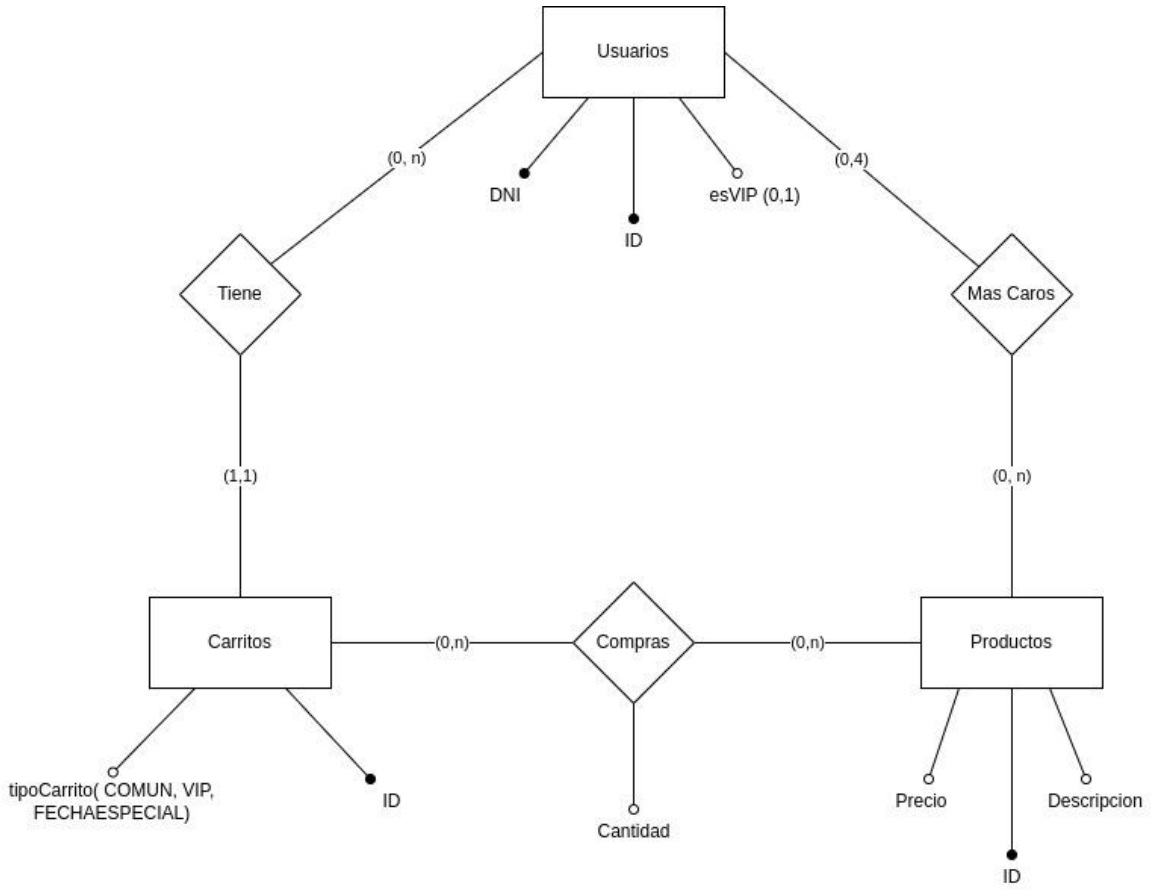
Al ver las características del caso de ("Los cuatro productos más caros"), se pueden tener en cuenta estos dos modelos

Propuesta 1 - Modelo teórico ideal - El utilizado:



Este modelo cumple con las características teóricas ideales, evitando bucles de información. Sin embargo, la búsqueda de los cuatro productos más caros de un cliente podría consumir una mayor cantidad de recursos computacionales.

Propuesta 2 - Modelo optimizado para consultas de usuario:



Este modelo agrega una relación "Más Caros" entre la tabla usuarios y la tabla productos. Esto ahorra tiempo computacional a la hora de obtener un usuario. Sin embargo, formaría un bucle, generando información redundante y empeorando el procesamiento en el caso de eliminar un carrito, eliminar item de un carrito, y aún peor en el caso no cubierto de eliminar un producto.

Quedaría determinar cuál sería la mejor opción a partir de la siguiente pregunta ¿Es más común solicitar los cuatro productos más caros, o eliminar de un carrito o el carrito entero?

Se considerará que la operación de eliminar un carrito es más utilizada. Además, se habló de un "ejercicio tipo facultad", así que se dará una solución académica, a pesar de ser la menos optimizada. Por lo que se tomará el primer modelo en base a esas premisas.

Tablas Utilizadas:

```
CREATE TABLE `usuarios` (
   `id` bigInt NOT NULL AUTO_INCREMENT ,
   `DNI` bigInt NOT NULL ,
   `esVIP` BOOLEAN NOT NULL DEFAULT FALSE ,
   PRIMARY KEY (`id`), UNIQUE (`DNI`)
);

CREATE TABLE `carritos` (
   `id` bigInt NOT NULL AUTO_INCREMENT,
   `tipoCarrito` enum('COMUN','VIP','FECHAESPECIAL') NOT NULL DEFAULT
'no',
   `idUsuario` bigInt NOT NULL,
   PRIMARY KEY (`id`),
   `idUsuario` (`idUsuario`),
   FOREIGN KEY (`idUsuario`) REFERENCES usuarios(id)
);

CREATE TABLE `productos` (
   `id` bigInt NOT NULL AUTO_INCREMENT,
   `descripcion` text NOT NULL,
   `precio` bigInt NOT NULL,
   PRIMARY KEY (`id`)
);

CREATE TABLE `compras` (
   `id` bigInt NOT NULL AUTO_INCREMENT ,
   `idCarrito` bigInt NOT NULL ,
   `idProducto` bigInt NOT NULL ,
   `cantidad` bigInt NOT NULL ,
   PRIMARY KEY (`id`),
   FOREIGN KEY (idCarrito) REFERENCES carritos(id),
   FOREIGN KEY (idProducto) REFERENCES productos(id)
);
```

Sobre comportamiento:

Fecha especial:

- Se asume que la fecha de promoción coincide con una propiedad del carrito, determinada por la fecha de su creación.

Dudas sobre comportamientos específicos:

- Se pide no modelar la compra, pero ¿Qué ocurre si el carrito solo es generado pero no es efectivamente comprado y pasa la fecha de promoción?
    - Podría agregarse una propiedad que haga efectivo el estado de esta propiedad en el momento en que sea efectivamente comprado.

Es VIP:

- Se asume que la promoción VIP coincide con una propiedad del carrito, determinada por el tipo de usuario al momento de su creación

Dudas sobre comportamientos específicos:

- Se pide no modelar la compra, pero ¿Qué ocurre si el carrito solo es generado pero no es efectivamente comprado y el usuario dejase de ser VIP?
    - Podría agregarse una propiedad que haga efectivo el estado de esta propiedad en el momento en que sea efectivamente comprado.

En ambos casos, el problema se solucionaría agregando el atributo "fueComprado" al carrito, el cual sería modificado al momento de su cierre al efectuar la compra, en vez de cuando se crea el carrito. Una vez cerrado el carrito, se hace efectivo el descuento.

Recomendaciones/Preguntas para el cliente de la aplicación:

Si se compran mas de 10 productos:

- Si el carrito es común se hará un descuento de 200$
    - ¿Qué ocurre si la compra es menor o igual a $200?

- Si el carrito es promocionable por fecha especial se hace un descuento general de 500 $.
    - ¿Qué ocurre si la compra es menor o igual a $500?

- Si el carrito es vip, se bonifica el producto más barato y se hace un descuento general de 700 $
    - ¿Qué ocurre si la compra es menor o igual a $700?

- No se tiene en cuenta si existe stock de un producto
    - ¿Qué ocurre si no hay más stock?
    - Se considera stock permanente para esta solucion.

Diseño de la aplicación

Se utiliza un modelo de tres capas

Capa de control (API Layer)
Procesa las solicitudes del cliente (GET POST PUT DELETE)

Capa de servicio (Business Logic)
Implementa la lógica del negocio.

Capa repositorio (Persistence Logic)
Capa de persistencia

Dependencias usadas

Se utiliza Java con Spring Boot y Maven, Postman para probar solicitudes y PHPMyAdmin para controlar el estado de la base de datos.

Las dependencias son: Spring Web, Spring Data JPA, MySQL Driver, Spring Boot Starter Validation y Lombok.

Configuración de "application.properties"

Propiedades para conectarse con la base de datos. Modificar según sea necesario.

```
spring.datasource.url=jdbc:mysql://localhost:3306/ems?useSSL=false
spring.datasource.username=root
spring.datasource.password=1111

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto=update
```

Se realizó el modelo, los controladores y las inyección de estos en cada uno.

Se pueden crear y eliminar carritos sin problemas.

Las solicitudes para los cuatro más caros, agregar a carrito y eliminar de carrito están incompletas. Las solicitudes modifican correctamente la base de datos, solo falta devolver la respuesta al completar la solicitud.