Cave à vins (Partie 06)

# TP06 du module 02 – Spring Data

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Création de requêtes spécifiques * Compléter les entités pour obtenir toutes les associations du projet |

|  |
| --- |
| **Durée estimée** |
| 2 à 3 heures |

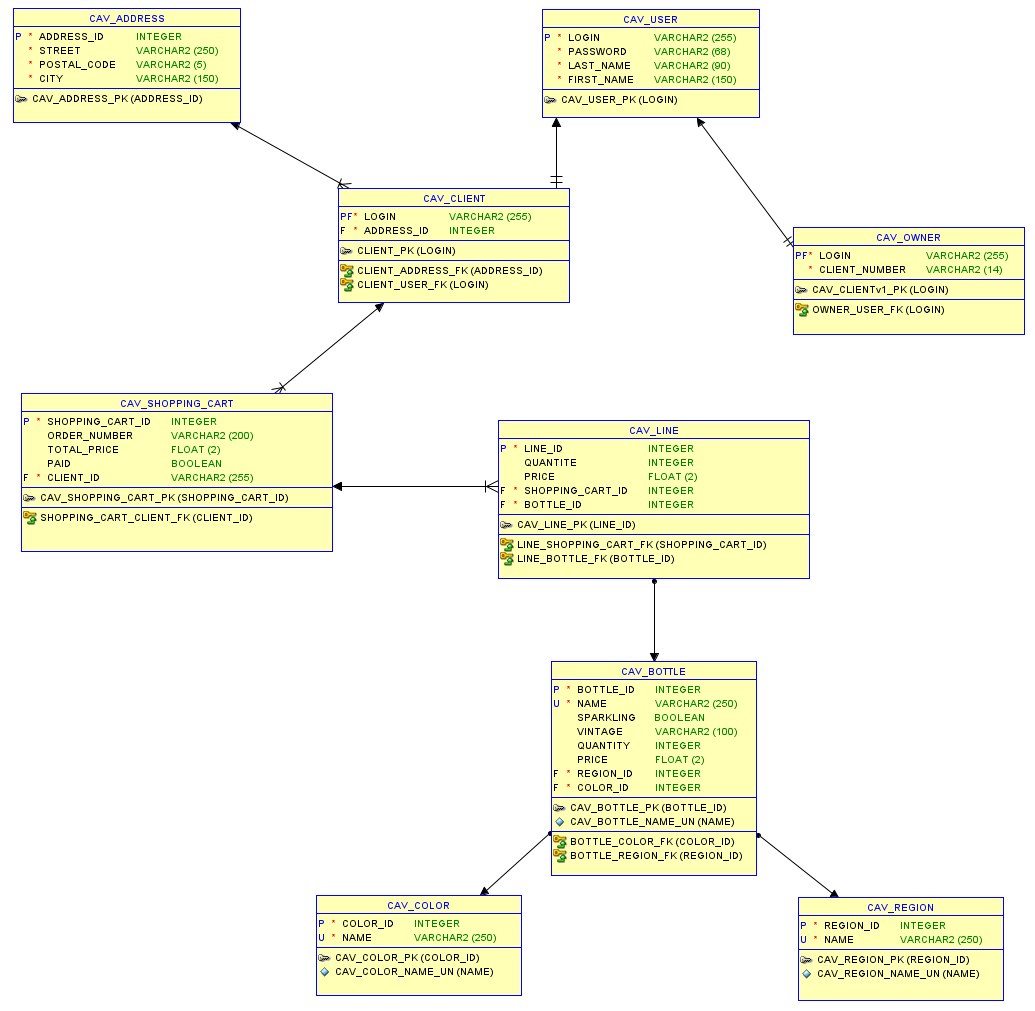
Solution

## Voici le diagramme complet des entités :

Une image contenant texte, diagramme, Parallèle, Plan

Description générée automatiquement

Voici le diagramme des tables en base de données :



## **Niveau 1** : Association ManyToOne LignePanier-Bouteille

1. Ajouter l’association ManyToOne entre LignePanier et Bouteille

Cette association est unidirectionnelle

* Il faut ajouter un attribut de Bouteille dans l’entité LignePanier
* Il n’y a pas de notion de cascade sur l’association
* Il faut préciser la colonne de jointure

**package** fr.eni.cave.bo.client;

**import** fr.eni.cave.bo.vin.Bouteille;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

//equals sur l'attribut id

@EqualsAndHashCode(of = { "id" })

@ToString

@Builder

@Entity

@Table(name = "CAV\_LINE")

**public** **class** LignePanier {

…

@ManyToOne

@JoinColumn(name = "BOTTLE\_ID")

**private** Bouteille bouteille;

}

1. Une classe de tests unitaires appelée TestManyToOneLignePanierBouteille vous est fournie :
   * Copier la classe dans le package fr.eni.cave.association des tests Java

* En vous aidant des tests unitaires des démonstrations et des TP précédents
  + Créer 2 tests unitaires :
  + Test\_save 🡪 sauvegarde une ligne associée à sa bouteille
  + Test\_delete 🡪 valide la suppression de la ligne mais la conservation de la bouteille

**package** fr.eni.cave.association;

**…**

@Slf4j

@DataJpaTest

**public** **class** TestManyToOneLignePanierBouteille {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

LignePanierRepository repository;

**private** Bouteille b1;

…

@Test

**public** **void** test\_save() {

**int** qte = 4;

**final** LignePanier lp = LignePanier

.*builder*()

.quantite(qte)

.prix(qte \* b1.getPrix())

.build();

// Association OneToMany

lp.setBouteille(b1);

// Appel du comportement

**final** LignePanier lpDB = repository.save(lp);

// Vérification de l'identifiant

*assertThat*(lpDB.getId()).isGreaterThan(0);

// Vérification de l'association

*assertThat*(lpDB.getBouteille()).isNotNull();

*assertThat*(lpDB.getBouteille().getId()).isEqualTo(b1.getId());

*assertThat*(lpDB.getPrix()).isEqualTo(qte \* b1.getPrix());

***log***.info(lpDB.toString());

}

@Test

**public** **void** test\_delete() {

**int** qte = 4;

**final** LignePanier lp = LignePanier

.*builder*()

.quantite(qte)

.prix(qte \* b1.getPrix())

.build();

// Association OneToMany

lp.setBouteille(b1);

// Appel du comportement

**final** LignePanier lpDB = entityManager.persist(lp);

entityManager.flush();

// Vérification de l'identifiant

*assertThat*(lpDB.getId()).isGreaterThan(0);

// Appel du comportement

repository.delete(lpDB);

// Vérification que l'entité a été supprimée

**final** LignePanier lpDB2 = entityManager.find(LignePanier.**class**, lpDB.getId());

*assertNull*(lpDB2);

//Vérifier que la bouteille n'a pas été supprimée

**final** Bouteille b = entityManager.find(Bouteille.**class**, b1.getId());

*assertNotNull*(b);

}

}

* Voici le genre de traces obtenu (les retours chariots et décalages sont là pour simplifier la lecture, ils ne font pas parti des traces d’origine)
  + Voici la ligne et sa bouteille :

LignePanier(

id=1,

quantite=4,

prix=95.8,

bouteille=Bouteille(id=1,

nom=DOMAINE ENI Ecole,

petillant=false,

millesime=2022,

quantite=1298, prix=23.95,

region=Region(id=1, nom=Pays de la Loire),

couleur=Couleur(id=1, nom=Blanc)))

## **Niveau 1** : Requêtes spécifiques

Dans notre application, il faut gérer des recherches spécifiques :

* Des bouteilles par une région
* Des bouteilles par une couleur
* Un utilisateur par son pseudo
* Un utilisateur par son pseudo et son mot de passe

1. Ces recherches doivent être des requêtes de Spring Data JPA avec mots clefs

* Compléter BouteilleRepository pour les recherches :
  + Des bouteilles par une région
  + Des bouteilles par une couleur

**package** fr.eni.cave.dal;

**import** java.util.List;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** org.springframework.data.repository.query.Param;

**import** fr.eni.cave.bo.vin.\*;

**public** **interface** BouteilleRepository **extends** JpaRepository<Bouteille, Integer> {

// Rechercher des bouteilles par leur région

List<Bouteille> findByRegion(@Param("r") Region r);

// Rechercher des bouteilles par leur couleur

List<Bouteille> findByCouleur(@Param("c") Couleur c);

}

* Compléter UtilisateurRepository pour les recherches :
  + Un utilisateur par son pseudo
  + Un utilisateur par son pseudo et son mot de passe

**package** fr.eni.cave.dal;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** org.springframework.data.repository.query.Param;

**import** fr.eni.cave.bo.Utilisateur;

**public** **interface** UtilisateurRepository **extends** JpaRepository<Utilisateur, String> {

//Rechercher un utilisateur par son pseudo

Utilisateur findByPseudo(@Param("pseudo") String pseudo);

//Rechercher un utilisateur par son pseudo et password

Utilisateur findByPseudoAndPassword(@Param("pseudo") String pseudo, @Param("password") String password);

}

1. Une classe de tests unitaires appelée TestRequetes vous est fournie
   * Copier la classe dans le package fr.eni.cave.jpql des tests Java

* Créer des tests unitaires pour valider chaque requête
  + Nous allons tester les cas viables des 4 méthodes
  + Et ajouter pour pseudo un cas inconnu
  + De même pour pseudo et password

**package** fr.eni.cave.jpql;

…

@DataJpaTest

**public** **class** TestRequetes {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

UtilisateurRepository utilisateurRepository;

@Autowired

BouteilleRepository bouteilleRepository;

Region paysDeLaLoire;

Couleur blanc;

List<Bouteille> bouteilles;

…

@Test

**public** **void** test\_BouteilleRepository\_findByRegion() {

**final** List<Bouteille> listeBouteilleDev = bouteilleRepository.findByRegion(paysDeLaLoire);

// Vérification

*assertThat*(listeBouteilleDev).isNotNull();

*assertThat*(listeBouteilleDev).isNotEmpty();

*assertThat*(listeBouteilleDev.size()).isEqualTo(3);

}

@Test

**public** **void** test\_BouteilleRepository\_findByCouleur() {

**final** List<Bouteille> listeBouteilleDev = bouteilleRepository.findByCouleur(blanc);

// Vérification

*assertThat*(listeBouteilleDev).isNotNull();

*assertThat*(listeBouteilleDev).isNotEmpty();

*assertThat*(listeBouteilleDev.size()).isEqualTo(2);

}

@Test

**public** **void** test\_UtilisateurRepository\_findByPseudo() {

String pseudo = "georgelucas@email.fr";

**final** Utilisateur utilisateur = utilisateurRepository.findByPseudo(pseudo);

// Vérification

*assertThat*(utilisateur).isNotNull();

*assertThat*(utilisateur.getPseudo()).isNotNull();

*assertThat*(utilisateur.getPseudo()).isEqualTo(pseudo);

}

@Test

**void** test\_UtilisateurRepository\_findByPseudo\_inconnu() {

String pseudo = "george";

**final** Utilisateur utilisateur = utilisateurRepository.findByPseudo(pseudo);

// Vérification

*assertThat*(utilisateur).isNull();

}

@Test

**public** **void** test\_UtilisateurRepository\_findByPseudoAndPassword() {

String pseudo = "georgelucas@email.fr";

String pwd = "Réalisateur&Producteur";

**final** Utilisateur utilisateur = utilisateurRepository.findByPseudoAndPassword(pseudo, pwd);

// Vérification

*assertThat*(utilisateur).isNotNull();

*assertThat*(utilisateur.getPseudo()).isNotNull();

*assertThat*(utilisateur.getPseudo()).isEqualTo(pseudo);

*assertThat*(utilisateur.getPassword()).isNotNull();

*assertThat*(utilisateur.getPassword()).isEqualTo(pwd);

}

@Test

**void** test\_UtilisateurRepository\_findByPseudoAndPassword\_faux() {

String pseudo = "georgelucas@email.fr";

String pwd = "Réalisateur&P";

**final** Utilisateur utilisateur = utilisateurRepository.findByPseudoAndPassword(pseudo, pwd);

// Vérification

*assertThat*(utilisateur).isNull();

}

}

## **Niveau 2** : Association ManyToOne Panier-Client

1. Ajouter l’association ManyToOne entre Panier et Client

* Il s’agit d’une association unidirectionnelle.
* Il suffit d’ajouter un Client dans le Panier

**package** fr.eni.cave.bo.client;

**import** java.util.ArrayList;

**import** java.util.List;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

//equals sur l'attribut id

@EqualsAndHashCode(of = { "id" })

@ToString

@Builder

@Entity

@Table(name = "CAV\_SHOPPING\_CART")

**public** **class** Panier {

…

// Association avec Client

@ManyToOne(cascade = { CascadeType.***PERSIST***, CascadeType.***MERGE***,

CascadeType.***REFRESH*** }, fetch = FetchType.***EAGER***)

@JoinColumn(name = "CLIENT\_ID")

**private** Client client;

}

* + Poser l’annotation @ManyToOne
  + Si le Client est créé, mis à jour ; il faut le sauver en même temps que le panier. Donc ajout du paramètre cascade : PERSIST, MERGE et REFRESH
  + Il est intéressant de le charger en même temps que son panier. Donc fetch a EAGER
  + Enfin, poser @JoinColumn pour préciser la colonne en table

1. Une classe de tests unitaires appelée TestManyToOnePanierClient vous est fournie
   * Copier la classe dans le package fr.eni.cave.association des tests Java

* En vous aidant des tests unitaires des démonstrations et des TP précédents
* Créer 3 tests unitaires :
  + test\_save\_1panier 🡪 sauvegarde un Panier et son Client associé
  + test\_save\_paniers\_unClient 🡪 sauvegarder plusieurs Panier d’un même Client
  + test\_delete 🡪 montrer que supprimer le Client ne supprime pas les Panier

**package** fr.eni.cave.association;

**…**

@Slf4j

@DataJpaTest

**public** **class** TestManyToOnePanierClient {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

PanierRepository panierRepository;

@Autowired

BouteilleRepository bouteilleRepository;

@Autowired

ClientRepository clientRepository;

…

@Test

**public** **void** test\_save\_1panier() {

**final** List<Bouteille> bouteilles = bouteilleRepository.findAll();

*assertThat*(bouteilles).isNotNull();

*assertThat*(bouteilles).isNotEmpty();

*assertThat*(bouteilles.size()).isEqualTo(2);

**final** Client client = Client

.*builder*()

.pseudo("bobeponge@email.fr")

.password("carré")

.nom("Eponge")

.prenom("Bob")

.adresse(Adresse

.*builder*()

.rue("Sous la mer")

.codePostal("35000")

.ville("Rennes")

.build())

.build();

**final** Panier panier = **new** Panier();

**final** Bouteille b1 = bouteilles.get(0);

**int** qte = 3;

**final** LignePanier lp1 = LignePanier

.*builder*()

.bouteille(b1)

.quantite(qte)

.prix(qte \* b1.getPrix())

.build();

panier.getLignesPanier().add(lp1);

panier.setPrixTotal(lp1.getPrix());

// Association ManyToOne

panier.setClient(client);

// Appel du comportement

**final** Panier panierDB = panierRepository.save(panier);

// Vérification de l'identifiant

*assertThat*(panierDB.getId()).isGreaterThan(0);

*assertThat*(panierDB.getClient()).isNotNull();

**final** Client clientDB = panierDB.getClient();

*assertThat*(clientDB.getPseudo()).isNotNull();

***log***.info(panierDB.toString());

}

@Test

**public** **void** test\_save\_paniers\_unClient() {

**final** Client client = Client

.*builder*()

.pseudo("bobeponge@email.fr")

.password("carré")

.nom("Eponge")

.prenom("Bob")

.adresse(Adresse

.*builder*()

.rue("Sous la mer")

.codePostal("35000")

.ville("Rennes")

.build())

.build();

// Association ManyToOne

**final** List<Panier> paniers = jeuDeDonnees();

paniers.forEach(p -> {

p.setClient(client);

});

// Contexte de la DB

paniers.forEach(p -> {

panierRepository.save(p);

*assertThat*(p.getId()).isGreaterThan(0);

});

// Appel du comportement

**final** Client clientDB = entityManager.find(Client.**class**, client.getPseudo());

// Vérification de l'identifiant de l'client

*assertThat*(clientDB).isNotNull();

***log***.info(paniers.toString());

}

@Test

**public** **void** test\_delete() {

**final** Client client = Client

.*builder*()

.pseudo("bobeponge@email.fr")

.password("carré")

.nom("Eponge")

.prenom("Bob")

.adresse(Adresse

.*builder*()

.rue("Sous la mer")

.codePostal("35000")

.ville("Rennes")

.build())

.build();

// Association ManyToOne

**final** List<Panier> paniers = jeuDeDonnees();

paniers.forEach(p -> {

p.setClient(client);

});

// Contexte de la DB

paniers.forEach(p -> {

entityManager.persist(p);

*assertThat*(p.getId()).isGreaterThan(0);

});

entityManager.flush();

// Récupération des identifiants des paniers pour le test de suppression

List<Integer> listeIdPanier = paniers

.stream()

.map(Panier::getId)

.collect(Collectors.*toList*());

*assertThat*(listeIdPanier).isNotNull();

*assertThat*(listeIdPanier).isNotEmpty();

*assertThat*(listeIdPanier.size()).isEqualTo(2);

// Appel du comportement

clientRepository.delete(client);

// Vérification que l'entité a été supprimée

**final** Client clientDB1 = entityManager.find(Client.**class**, client.getPseudo());

*assertNull*(clientDB1);

// Les entités Panier doivent être en base

// Il n'y a pas la notion de DELETE sur notre relation

*assertThat*(listeIdPanier).isNotNull();

*assertThat*(listeIdPanier).isNotEmpty();

listeIdPanier.forEach(id -> {

*assertThat*(id).isGreaterThan(0);

Panier panierDB = entityManager.find(Panier.**class**, id);

*assertNotNull*(panierDB);

});

}

**private** List<Panier> jeuDeDonnees() {

**…**

}

}

* Voici le genre de traces à obtenir (les retours chariots et décalages sont là pour simplifier la lecture, ils ne font pas parti des traces d’origine)
  + Voici pour un Panier et son Client :

Panier(id=1, numCommande=null, prixTotal=34.35, paye=false,

lignesPanier=[LignePanier(id=1, quantite=3, prix=34.35,

bouteille=Bouteille(id=1, nom=DOMAINE ENI Ecole,

petillant=false, millesime=2022,

quantite=1298, prix=11.45,

region=Region(id=1, nom=Pays de la Loire),

couleur=Couleur(id=1, nom=Blanc)))]

* + Voici pour 2 Panier et le même Client :

[ Panier(id=1, numCommande=null, prixTotal=34.35, paye=false,

lignesPanier=[LignePanier(id=1, quantite=3, prix=34.35,

bouteille=Bouteille(id=1, nom=DOMAINE ENI Ecole,

petillant=false, millesime=2022,

quantite=1298, prix=11.45,

region=Region(id=1, nom=Pays de la Loire),

couleur=Couleur(id=1, nom=Blanc)))],

client=Utilisateur(pseudo=bobeponge@email.fr, nom=Eponge, prenom=Bob)),

Panier(id=2, numCommande=null, prixTotal=239.5, paye=false,

lignesPanier=[LignePanier(id=2, quantite=10, prix=239.5,

bouteille=Bouteille(id=2, nom=DOMAINE ENI Service,

petillant=false, millesime=2015,

quantite=2998, prix=23.95,

region=Region(id=3, nom=Nouvelle-Aquitaine),

couleur=Couleur(id=2, nom=Rouge)))],

client=Utilisateur(pseudo=bobeponge@email.fr, nom=Eponge, prenom=Bob))]

## **Niveau 2** : Requêtes spécifiques

Dans notre application, il faut ajouter des recherches plus spécifiques :

* Les paniers d’un client (les paniers qui ont un numéro de commande nul)
* Les commandes d’un client (les paniers qui ont un numéro de commande non nul)

1. Déterminer 2 solutions pour la première recherche :
   * 1 solution avec une requête JPQL
   * 1 solution avec une requête Spring Data JPA par mots clefs

* Dans PanierRepository, création des 2 méthodes

**package** fr.eni.cave.dal;

**import** java.util.List;

**import** org.springframework.data.jpa.repository.\*;

**import** org.springframework.data.repository.query.Param;

**import** fr.eni.cave.bo.client.Client;

**import** fr.eni.cave.bo.client.Panier;

**public** **interface** PanierRepository **extends** JpaRepository<Panier, Integer> {

// Création d'une requête paramétrée avec JPQL

@Query("SELECT p FROM Panier p WHERE p.client = :client AND p.numCommande = null")

List<Panier> findPaniersWithJPQL(@Param("client") Client client);

// Rechercher la liste des paniers non commandés d'un client

List<Panier> findByNumCommandeNullAndClient(@Param("client") Client client);

}

1. Déterminer 2 solutions pour la seconde recherche :
   * 1 solution avec une requête native
   * 1 solution avec une requête Spring Data JPA par mots clefs

* Dans PanierRepository, création des 2 méthodes
  + Pour le SQL, le paramètre sera uniquement le pseudo du client

…

// Création d'une requête native - SQL

@Query(value = "SELECT p.\* FROM CAV\_SHOPPING\_CART p WHERE p.CLIENT\_ID = :idClient AND p.ORDER\_NUMBER IS NOT NULL", nativeQuery = **true**)

List<Panier> findCommandesWithSQL(@Param("idClient")String idClient);

// Rechercher la liste des commandes d'un client

List<Panier> findByNumCommandeNotNullAndClient(@Param("client") Client client);

}

1. Une classe de tests unitaires appelée TestRequetesPlusPoussees vous est fournie
   * Copier la classe dans le package fr.eni.cave.jpql des tests Java

* Créer des tests unitaires pour valider chaque requête
  + Nous testons les 4 méthodes avec 1 client des paniers actifs et des commandes

**package** fr.eni.cave.jpql;

**import** **static** org.assertj.core.api.Assertions.*assertThat*;

**import** java.util.\*;

**import** org.junit.jupiter.api.\*;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.test.autoconfigure.orm.jpa.\*;

**import** fr.eni.cave.bo.client.\*;

**import** fr.eni.cave.bo.vin.\*;

**import** fr.eni.cave.dal.\*;

@DataJpaTest

**public** **class** TestRequetesPlusPoussees {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

BouteilleRepository bouteilleRepository;

Region paysDeLaLoire;

Couleur blanc;

List<Bouteille> bouteilles;

Client tom;

@Autowired

PanierRepository panierRepository;

…

@Test

**void** test\_utilisateurRepository\_findPaniersWithJPQL() {

**final** List<Panier> paniers = panierRepository.findPaniersWithJPQL(tom);

// Vérification

*assertThat*(paniers).isNotNull();

*assertThat*(paniers).isNotEmpty();

*assertThat*(paniers.size()).isEqualTo(1);

Panier p = paniers.get(0);

*assertThat*(p).isNotNull();

*assertThat*(p.getNumCommande()).isNull();

}

@Test

**void** test\_utilisateurRepository\_findByNumCommandeNullAndClient() {

**final** List<Panier> paniers = panierRepository.findByNumCommandeNullAndClient(tom);

// Vérification

*assertThat*(paniers).isNotNull();

*assertThat*(paniers).isNotEmpty();

*assertThat*(paniers.size()).isEqualTo(1);

Panier p = paniers.get(0);

*assertThat*(p).isNotNull();

*assertThat*(p.getNumCommande()).isNull();

}

@Test

**void** test\_utilisateurRepository\_findCommandesWithSQL() {

**final** List<Panier> commandes = panierRepository.findCommandesWithSQL(tom.getPseudo());

// Vérification

*assertThat*(commandes).isNotNull();

*assertThat*(commandes).isNotEmpty();

*assertThat*(commandes.size()).isEqualTo(2);

commandes.forEach(c -> {

String numCommande = tom.getPseudo() + "\_" + c.getId();

*assertThat*(c.getNumCommande()).isNotNull();

*assertThat*(c.getNumCommande()).isEqualTo(numCommande);

});

}

@Test

**void** test\_utilisateurRepository\_findByNumCommandeNotNullAndClient() {

**final** List<Panier> commandes = panierRepository.findByNumCommandeNotNullAndClient(tom);

// Vérification

*assertThat*(commandes).isNotNull();

*assertThat*(commandes).isNotEmpty();

*assertThat*(commandes.size()).isEqualTo(2);

commandes.forEach(c -> {

String numCommande = tom.getPseudo() + "\_" + c.getId();

*assertThat*(c.getNumCommande()).isNotNull();

*assertThat*(c.getNumCommande()).isEqualTo(numCommande);

});

}

}