JPQL et méthodes de requêtes

# Démonstration 14 du module 2

|  |
| --- |
| Les objectifs de cette démonstration sont :   * Déclaration de méthodes de requêtes dans les Repository * Utilisation de JPQL pour définir des requêtes * Création de requête native en SQL |

Déroulement

# Contexte

* Continuer l’application précédente
* Dans cette itération, nous allons utiliser Spring Data JPA pour créer des requêtes de recherches particulières
  + Avec des méthodes de requêtes
  + JPQL
  + Voir SQL
* Dans nos itérations précédentes, nous avions laissé des TODO
  + Ils correspondent à la recherche d’un employé par son immatriculation.
  + Nous allons pouvoir compléter ces TODO

# Dans EmployeRepository :

* Création d’une méthode de requêtes avec les mots clefs pour rechercher un employé par son immatriculation :

Employe findByImmatriculation(String immatriculation);

* Création d’une méthode annotée @Query pour rechercher un employé par son email (utilisation du JPQL) :

@Query("SELECT e FROM Employe e WHERE e.email = :email")

Employe findByEmailWithJPQL(@Param("email") String email);

* + Il serait possible de créer cette requête avec la solution d’une méthode de requêtes avec les mots clefs ; cela donnerait :

Employe findByEmail(String email);

* Création d’une méthode de requêtes avec les mots clefs pour remonter l’ensemble des employés trié par leur civilité :

List<Employe> findByOrderByCivilite();

**package** fr.eni.demo.dal;

**import** java.util.List;

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

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

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

**import** fr.eni.demo.bo.Employe;

**public** **interface** EmployeRepository **extends** JpaRepository<Employe, Integer> {

// Création d'une requête paramétrée avec JPQL

@Query("SELECT e FROM Employe e WHERE e.email = :email")

Employe findByEmailWithJPQL(@Param("email") String email);

// Méthodes de requêtes avec mots clefs via Spring Data JPA

Employe findByImmatriculation((@Param("immatriculation") String immatriculation);

// Méthodes de requêtes avec mots clefs via Spring Data JPA

List<Employe> findByOrderByCivilite();

}

# Dans CoursRepository :

* Création d’une méthode annotée @Query pour rechercher tous les cours en filière développement (utilisation du JPQL) :

@Query("SELECT c FROM Cours c WHERE c.filiere = 'Développement'")

List<Cours> findWithJPQL();

* + Nous manipulons les classes et les attributs
  + Cours et filiere
* Création d’une méthode annotée @Query et le paramètre nativeQuery pour rechercher un employé par son email en utilisant du SQL :

@Query(value = "SELECT c.\* FROM COMPUTER\_COURSE c WHERE c.COMPUTER\_SCIENCE\_COURSE like '%Réseau%'", nativeQuery = **true**)

List<Cours> findWithSQL();

* + Nous manipulons la base de données en SQL
  + D’où la table COMPUTER\_COURSE
  + Et la colonne COMPUTER\_SCIENCE\_COURSE

**package** fr.eni.demo.dal.formation;

**import** java.util.List;

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

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

**import** fr.eni.demo.bo.formation.Cours;

**public** **interface** CoursRepository **extends** JpaRepository<Cours, Integer> {

// Création d'une requête avec JPQL

@Query("SELECT c FROM Cours c WHERE c.filiere = 'Développement'")

List<Cours> findWithJPQL();

// Création d'une requête native - SQL

@Query(

value = "SELECT c.\* FROM COMPUTER\_COURSE c WHERE c.COMPUTER\_SCIENCE\_COURSE like '%Réseau%'",

nativeQuery = **true**)

List<Cours> findWithSQL();

}

# Utilisation des tests unitaires

* Créer une classe TestJPQLMethodesRequetes
  + Se faire injecter TestEntityManager, EmployeRepository et CiviliteRepository
* Créer une méthode de pré-chargement, elle va charger :
  + Des civilités
  + Des employés
  + Des cours
* Créer un test par méthode à valider :

**package** fr.eni.demo.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.demo.bo.\*;

**import** fr.eni.demo.dal.EmployeRepository;

**import** fr.eni.demo.dal.formation.CoursRepository;

**import** lombok.extern.slf4j.Slf4j;

@Slf4j

@DataJpaTest

**public** **class** TestJPQLMethodesRequetes {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

EmployeRepository employeRepository;

@Autowired

CoursRepository coursRepository;

@BeforeEach

**public** **void** jeuDeDonnees() {

Civilite monsieur = Civilite

.*builder*()

.clef("M")

.libelle("Monsieur")

.build();

Civilite madame = Civilite

.*builder*()

.clef("Mme")

.libelle("Madame")

.build();

Civilite mx = Civilite

.*builder*()

.clef("Mx")

.libelle("Mix")

.build();

entityManager.persist(monsieur);

entityManager.persist(madame);

entityManager.persist(mx);

List<Employe> employes = **new** ArrayList<>();

employes.add(Employe

.*builder*()

.civilite(monsieur)

.nom("NICOLAS")

.prenom("Cédric")

.email("cnicolas@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_10100")

.numDom("02XXXXXXXX")

.numPortable("06XXXXXXXX")

.build());

employes.add(Formateur

.*builder*()

.civilite(madame)

.nom("BAILLE")

.prenom("Anne-Lise")

.email("abaille@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.filiere("Développement")

.build());

employes.add(Formateur

.*builder*()

.civilite(mx)

.nom("DELACHESNAIS")

.prenom("Frédéric")

.email("fdelachesnais@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_14398")

.numDom("02XXXXXXXX")

.numPortable("06XXXXXXXX")

.filiere("Développement")

.build());

employes.add(Formateur

.*builder*()

.civilite(monsieur)

.nom("MATHIEU")

.prenom("Wilfrid")

.email("wmathieu@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_14388")

.numDom("02XXXXXXXX")

.numPortable("06XXXXXXXX")

.filiere("Système et Réseau")

.build());

employes.add(ChargeRelationsEcoleEntreprises

.*builder*()

.civilite(mx)

.nom("PAGEOT")

.prenom("Emma")

.email("epageot@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_13398")

.numDom("02XXXXXXXX")

.numPortable("06XXXXXXXX")

.numeroBureau("04XXXXXXXX")

.build());

employes.forEach(e -> {

entityManager.persist(e);

// Vérification de l'identifiant

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

});

List<Cours> listeCoursDB = **new** ArrayList<>();

listeCoursDB.add(Cours

.*builder*()

.filiere("Développement")

.reference("DEV2023-M030")

.titre("Web Client HTML CSS")

.duree(5)

.build());

listeCoursDB.add(Cours

.*builder*()

.filiere("Développement")

.reference("DEV2023-M310")

.titre("Framework JS Angular")

.duree(5)

.build());

listeCoursDB.add(Cours

.*builder*()

.filiere("Développement")

.reference("DEV2023-M360")

.titre("Java Frameworks - APIs Web")

.duree(10)

.build());

listeCoursDB.add(Cours

.*builder*()

.filiere("Système et Réseau")

.reference("TSRR-120")

.titre("Bases des réseaux")

.duree(5)

.build());

listeCoursDB.add(Cours

.*builder*()

.filiere("Système et Réseau")

.reference("TSRR-170")

.titre("GLPI")

.duree(5)

.build());

listeCoursDB.forEach(c -> {

entityManager.persist(c);

// Vérification de l'identifiant

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

});

}

@Test

**public** **void** test\_requete\_JPQL() {

**final** List<Cours> listeCoursDev = coursRepository.findWithJPQL();

// Vérification

*assertThat*(listeCoursDev).isNotNull();

*assertThat*(listeCoursDev).isNotEmpty();

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

}

@Test

**public** **void** test\_requete\_JPQL\_parametree() {

**final** String email = "epageot@campus-eni.fr";

**final** Employe employe = employeRepository.findByEmailWithJPQL(email);

// Vérification

*assertThat*(employe).isNotNull();

*assertThat*(employe.getNom()).isNotNull();

*assertThat*(employe.getNom()).isEqualTo("PAGEOT");

}

@Test

**public** **void** test\_methode\_requete\_findByImmatriculation() {

**final** String immatriculation = "ENI\_ECOLE\_14388";

**final** Employe employe = employeRepository.findByImmatriculation(immatriculation);

// Vérification

*assertThat*(employe).isNotNull();

*assertThat*(employe.getNom()).isNotNull();

*assertThat*(employe.getNom()).isEqualTo("MATHIEU");

}

@Test

**public** **void** test\_methode\_requete\_findByOrderByCivilite() {

**final** List<Employe> employes = employeRepository.findByOrderByCivilite();

// Vérification

*assertThat*(employes).isNotNull();

*assertThat*(employes).isNotEmpty();

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

}

@Test

**public** **void** test\_requete\_native\_SQL() {

**final** List<Cours> coursReseaux = coursRepository.findWithSQL();

// Vérification

*assertThat*(coursReseaux).isNotNull();

*assertThat*(coursReseaux).isNotEmpty();

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

}

}

* Tous les tests doivent être verts
* Pour la méthode findByOrderByCivilite, nous obtenons le tri :

[

Employe(id=1,

civilite=Civilite(clef=M, libelle=Monsieur),

nom=NICOLAS, prenom=Cédric, email=cnicolas@campus-eni.fr,

immatriculation=ENI\_ECOLE\_10100, numDom=02XXXXXXXX, numPortable=06XXXXXXXX, adresse=null),

Formateur(

super=Employe

(id=4,

civilite=Civilite(clef=M, libelle=Monsieur),

nom=MATHIEU, prenom=Wilfrid, email=wmathieu@campus-eni.fr,

immatriculation=ENI\_ECOLE\_14388, numDom=02XXXXXXXX, numPortable=06XXXXXXXX, adresse=null),

filiere=Système et Réseau),

Formateur(

super=Employe(id=2,

civilite=Civilite(clef=Mme, libelle=Madame),

nom=BAILLE, prenom=Anne-Lise, email=abaille@campus-eni.fr,

immatriculation=ENI\_ECOLE\_12398, numDom=02XXXXXXXX, numPortable=null, adresse=null),

filiere=Développement),

Formateur(

super=Employe(id=3,

civilite=Civilite(clef=Mx, libelle=Mix),

nom=DELACHESNAIS, prenom=Frédéric, email=fdelachesnais@campus-eni.fr,

immatriculation=ENI\_ECOLE\_14398, numDom=02XXXXXXXX, numPortable=06XXXXXXXX, adresse=null),

filiere=Développement),

ChargeRelationsEcoleEntreprises(

super=Employe(id=5,

civilite=Civilite(clef=Mx, libelle=Mix),

nom=PAGEOT, prenom=Emma, email=epageot@campus-eni.fr, immatriculation=ENI\_ECOLE\_13398,

numDom=02XXXXXXXX, numPortable=06XXXXXXXX, adresse=null), numeroBureau=04XXXXXXXX)

]

* + Les employés sont bien triés selon leur civilité et l’ordre alphabétique

# TODO - BLL

* Supprimons les TODO
  + Utiliser l’onglet Tasks pour les retrouver
* Modifier la méthode validerImmatriculation de EmployeServiceImpl

**private** **void** validerImmatriculation(Employe employe) {

// Valider que l'immatriculation n'est pas nule ou vide

validerChaineNonNulle(employe.getImmatriculation(), "L'immatriculation n'a pas été renseignée");

// Immatriculation doit être unique

// Appel de la méthode de requête spécifique : findByImmatriculation

Employe employeDB = employeRepository.findByImmatriculation(employe.getImmatriculation());

**if** (employeDB != **null**) {

**throw** **new** RuntimeException("L'immatriculation doit être unique");

}

}

* Compléter la méthode test\_ajouter\_immatriculationUnique de tests de la classe TestEmployeService

// Immatriculation Unique - la validation métier l'interdit

@Test

**void** test\_ajouter\_immatriculationUnique() {

Employe employe1 = Employe

.*builder*()

.id(1)

.nom("BAILLE")

.prenom("Anne-Lise")

.email("abaille@campus-eni.fr")

.immatriculation("ENI\_Ecole\_012892")

.numDom("0299XXXXXX")

.numPortable("0699XXXXXX")

.build();

employeService.ajouter(employe1);

// Définir le comportement du Repository avec findByImmatriculation

// Méthode de requête : findByImmatriculation

*when*(employeRepository.findByImmatriculation("ENI\_Ecole\_012892")).thenReturn(employe1);

Employe employe2 = Employe

.*builder*()

.id(2)

.nom("GOBIN")

.prenom("Stephane")

.email("sgobin@campus-eni.fr")

.immatriculation("ENI\_Ecole\_012892")

.numDom("0288XXXXXX")

.build();

*assertThrows*(RuntimeException.**class**, () -> employeService.ajouter(employe2));

}

* Relancer le test unitaire, il est valide à présent

Les recherches spécifiques se font par méthodes de requêtes ou via JPQL. Ainsi, nous conservons les avantages de Spring Data JPA et ouvrons à des recherches supplémentaires