Clefs primaires composites (méthode 1)

# Démonstration 3 du module 2

|  |
| --- |
| L’objectif de cette démonstration :   * Déclaration d’une clef primaire composite |

Déroulement

# Contexte

* Continuer l’application précédente
* Dans cette itération, nous allons ajouter la notion de stagiaire :
  + Voici la table en base de données désirées

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* + Un stagiaire a
    - Une clef primaire composée de son email ENI et son numéro d’étudiant ENI
    - Un nom et un prénom obligatoires
    - Un numéro de téléphone de domicile et/ou de portable
    - Un email personnel obligatoire

# Création de la classe gérant la clé primaire :

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

**import** java.io.Serializable;

**import** lombok.\*;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

**public** **class** EtudiantPK **implements** Serializable{

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String email;

**private** String immatriculation;

}

* Elle déclare les 2 attributs de la clef primaire
* Il faut qu’elle respecte le design pattern POJO
  + Pour cela ajout des annotations de Lombok :
  + @Data
  + @NoArgsConstructor
  + @AllArgsConstructor
  + @Builder
* Elle doit implémenter Serializable pour être embarquée

# Création de la classe d’entité (BO) :

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

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

**import** lombok.\*;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

/\*\*

\*

\* **@author** Eni Ecole

\*

\* **@category** Voici une entité avec une clef composite correspondant à l'email et

\* l'immatriculation

\*/

@Entity

@Table(name = "STUDENT")

//Définir la classe de la clef composite

@IdClass(EtudiantPK.**class**)

**public** **class** Etudiant {

@Id

@Column(nullable = **false**, unique = **true**, length = 255)

**private** String email;

@Id

@Column(name = "STUDENT\_REGISTRATION", nullable = **false**, unique = **true**, length = 100)

**private** String immatriculation;

@Column(name = "LAST\_NAME", nullable = **false**, length = 90)

**private** String nom;

@Column(name = "FIRST\_NAME", nullable = **false**, length = 150)

**private** String prenom;

@Column(name = "HOME\_PHONE\_NUMBER", length = 12)

**private** String numDom;

@Column(name = "CELL\_NUMBER", length = 12)

**private** String numPortable;

@Column(name="PERSONAL\_EMAIL", nullable = **false**, unique = **true**, length = 255)

**private** String emailPersonnel;

}

* Cette classe est très similaire à la classe Employe
* Les différences sont
  + L’annotation @Id est placée sur 2 attributs :
    - email
    - immatriculation
  + Ajout de l’annotation @IdClass pour préciser la classe de la clef composite

# Création de EtudiantRepository

* Créer l’interface EtudiantRepository qui hérite de JpaRepository<Etudiant, EtudiantPK>
* La complexité est de bien faire attention d’utiliser la classe de la clef composite comme classe pour l’identifiant

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

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

**import** fr.eni.demo.bo.pk.Etudiant;

**import** fr.eni.demo.bo.pk.EtudiantPK;

**public** **interface** EtudiantRepository **extends** JpaRepository<Etudiant, EtudiantPK>{

}

# Création d’un test unitaire pour valider la clef composite

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

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

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

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

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

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

**import** fr.eni.demo.bo.pk.Etudiant;

**import** fr.eni.demo.bo.pk.EtudiantPK;

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

@Slf4j

@DataJpaTest

**public** **class** TestEtudiantRepository {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

EtudiantRepository repository;

@Test

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

**final** Etudiant entiteASauver = Etudiant

.*builder*()

.nom("SOPRANO")

.prenom("Camille")

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

.immatriculation("ENI\_CAMPUS\_202311872")

.numDom("02XXXXXXXX")

.emailPersonnel("camille.soprano@perso.fr")

.build();

// Appel du comportement

**final** Etudiant entite = repository.save(entiteASauver);

*assertThat*(entite).isNotNull();

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

**final** EtudiantPK pk = EtudiantPK

.*builder*()

.email(entiteASauver.getEmail())

.immatriculation(entiteASauver.getImmatriculation())

.build();

**final** Etudiant entiteDB = entityManager.find(Etudiant.**class**, pk);

*assertThat*(entiteDB).isNotNull();

*assertThat*(entite).isEqualTo(entiteDB);

}

}

## Exécution :

* Spring Data JPA a créé le script SQL suivant pour la génération de la table :

create table student (

cell\_number varchar(12),

home\_phone\_number varchar(12),

last\_name varchar(90) not null,

student\_registration varchar(100) not null unique,

first\_name varchar(150) not null,

email varchar(255) not null unique,

personal\_email varchar(255) not null unique,

primary key (student\_registration, email))

* Au niveau des traces :

Etudiant(email=csoprano@campus-eni.fr, immatriculation=ENI\_CAMPUS\_202311872, nom=SOPRANO, prenom=Camille, numDom=02XXXXXXXX, numPortable=null, emailPersonnel=camille.soprano@perso.fr)