Explications liées au programme DemoTest9JPAMultitables

<dependency> **<groupId>** **org.springframework.boot** **</groupId>** **<artifactId>** **spring-boot-starter-data-jpa** **</artifactId>** **</dependency>** **<dependency>** **<groupId>** **org.apache.derby** **</groupId>** **<artifactId>derbyclient</artifactId>** **</dependency>**	Fichier Pom → il faut ajouter les dépendances à Derby et à JPA

spring.datasource.url=jdbc:derby://localhost:1527/BcCliArt **spring.datasource.username=app** **spring.datasource.password=app** **spring.datasource.driver-class-name=org.apache.derby.jdbc.ClientDriver** **spring.jpa.hibernate.ddl-auto=update** **spring.datasource.initialization-mode=always**	Fichier **application.properties** : Permet de configurer la BD utilisée

<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"> **<persistence-unit** name="com.example_demo10_Derby_MultiTables_jar_0.0.1-SNAPSHOTPU" transaction-type="RESOURCE_LOCAL"**>** <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider> <class>tse.isib.demoTest9JpaMultitables.entity.Bcart</class> <class>tse.isib.demoTest9JpaMultitables.entity.Bc</class> <class>tse.isib.demoTest9JpaMultitables.entity.Article</class> <class>tse.isib.demoTest9JpaMultitables.entity.Client</class> <properties> <property name="javax.persistence.jdbc.url" value="jdbc:derby://localhost:1527/BcCliArt"/> <property name="javax.persistence.jdbc.user" value="app"/> <property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.ClientDriver"/> <property name="javax.persistence.jdbc.password" value="app"/> </properties> **</persistence-unit>** </persistence>	Fichier persistence.xml créé automatiquement par Netbeans en exécutant **« Entity Classes From Database »** à partir d'une connexion existante à la BD

@Entity
@Table(name = "CLIENT")
@XmlRootElement
@Data
@NoArgsConstructor
@AllArgsConstructor
 	
public class Client implements Serializable {
  private static final long serialVersionUID = 1L;
  @Id
  @Basic(optional = false)
  @Column(name = "NCLI")
  private String ncli;
  @Basic(optional = false)
  @Column(name = "NOM")
  private String nom;
  @Basic(optional = false)
  @Column(name = "PRENOM")
  private String prenom;
  @Basic(optional = false)
  @Column(name = "SEXE")
  private String sexe;
  @Basic(optional = false)
  @Column(name = "LOC")
  private String loc;
  @Basic(optional = false)
  @Column(name = "PASSWD")
  private String passwd;
  @OneToMany(cascade = CascadeType.ALL,
  mappedBy = "ncli", fetch=FetchType.EAGER)
  private Collection<Bc> bcCollection;
 	
}
	Les différents entities sont créés par NetBean
	
	L'Entity Client possède une relation 1 :N avec la table BC

@Entity
@Table(name = "ARTICLE")
@XmlRootElement
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Article implements Serializable {
 	
private static final long serialVersionUID = 1L;
@Id
@Basic(optional = false)
@Column(name = "NART")
private String nart;
@Basic(optional = false)
@Column(name = "LIBELLE")
private String libelle;
@Basic(optional = false)
@Column(name = "PU")
private int pu;
@Column(name = "CHEMIN")
private String chemin;
@OneToMany(cascade = CascadeType.ALL, mappedBy = "article",
fetch=FetchType.EAGER)
private Collection<Bcart> bcartCollection;
 	
}
	L'Entity Article possède une relation 1 :N avec la table BC

@Entity
@Table(name = "BC")
@XmlRootElement
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Bc implements Serializable {
  private static final long serialVersionUID = 1L;
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Basic(optional = false)
  @Column(name = "NBON")
  private Integer nbon;
  @Basic(optional = false)
  @Column(name = "LIBELLE")
  private String libelle;
  @JoinColumn(name = "NCLI",
  referencedColumnName = "NCLI")
  @ManyToOne
  private Client ncli;
  @OneToMany(cascade = CascadeType.ALL,
  mappedBy = "bc")
  private Collection<Bcart> bcartCollection;
 	
}
 	
	Entity BC
	
	• Relation N :1 avec la table Client
	
	• Relation 1 :N avec l'Entity de jointure Bcart

@Data
@NoArgsConstructor
@AllArgsConstructor
@Embeddable
 	
public class BcartPK implements Serializable {
  @Basic(optional = false)
  @Column(name = "NBON")
  private int nbon;
  @Basic(optional = false)
  @Column(name = "NART")
  private String nart;
 	
}
	Classe @Embeddable qui permet de définir une clé primaire composée pour l'Entity de jointure Bcart

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "BCART")
@XmlRootElement
public class Bcart implements Serializable {
 	
private static final long serialVersionUID = 1L;
@EmbeddedId
protected BcartPK bcartPK;
@Basic(optional = false)
@Column(name = "Q")
private int q;
@JoinColumn(name = "NART", referencedColumnName = "NART")
@ManyToOne(optional = false)
private Article article;
 	
@JoinColumn(name = "NBON", referencedColumnName = "NBON")
@ManyToOne(optional = false)
private Bc bc;
 	
}
	Entity Bcart → table de jointure entre Article et Bc

public interface ArticleDB extends CrudRepository<Article,String>
{
}
 	
public interface BcDB extends CrudRepository<Bc,Integer>
{
}
 	
public interface BcartDB extends CrudRepository<Bcart,BcartPK>
{
}
 	
public interface ClientDB extends CrudRepository<Client,String>
{
}
	Interfaces « **CrudRepository** » proposant des méthodes CRUD aux différentes tables de la BD

Exemples d'accès à la BD Multitables :

```
public class BdAcces implements CommandLineRunner {
   @Autowired
   private  ClientDB clientDB;
   @Autowired
   private  BcDB bcDB;
   @Autowired
   private  ArticleDB articleDB;

   @Autowired BcartDB bcArt;
   @Override
   public void run(String... strings) throws Exception {

     Iterable<Client> i=this.clientDB.findAll();  // On va chercher tous les clients
      System.out.println("Results:");
      for(Client C:i)
      {  System.out.println(C.getNom()+" "+C.getPrenom());
        Collection<Bc> col=C.getBcCollection();   // On récupère tous les BC du client traité
        for(Bc bc: col)
         { System.out.println(bc.getNbon()+" "+bc.getLibelle());
           Collection<Bcart> col2=bc.getBcartCollection();
                  // Pour chaque BC, on récupère les enregistrements de jointure
           for(Bcart bcart:col2 )
            {System.out.println("Article n°"+bcart.getBcartPK().getNart());
             Optional<Article> optArt=articleDB.findById(bcart.getBcartPK().getNart());
              System.out.println(optArt.get().getLibelle());
           }
         }
      }
```

```
Bc bon=new Bc();
// Création d'un nouveau bon de commande qui sera associé à différents articles
bon.setLibelle("Test creation bon");
Optional<Client> optCli=clientDB.findById("2");
bon.setNcli(optCli.get());     // On associe à ce bon un client existant
bcDB.save(bon);

int nbon=bon.getNbon();  // On accède au Nbon créé automatiquement par la BD
System.out.println("nbon:"+nbon);

 Collection<Bcart> bcCol=new ArrayList<>();
 Bcart bcart=new Bcart(nbon, "1");
 bcart.setBc(bon);
 Optional<Article> optArt=articleDB.findById("1");
 bcart.setArticle(optArt.get());
 bcart.setQ(5);
 bcCol.add(bcart);

 bcart=new Bcart(nbon, "3");
 bcart.setBc(bon);
 Optional<Article> optArt2=articleDB.findById("3");
 bcart.setArticle(optArt2.get());
 bcart.setQ(3);
 bcCol.add(bcart);

 // Ajouter d'autres articles eventuellement ....
 bon.setBcartCollection(bcCol);
 bcDB.save(bon);
// Le bon est associé à la liste de Bcart qui associe ce bon à des articles commandés
```

```
        Optional<Bc> optBc=bcDB.findById(6);
        bcDB.deleteById(optBc.get().getNbon());
       //  bcDB.delete(optBc.get());

    }
}
```