

Cours INF3522 - Développement d'Applications N tier

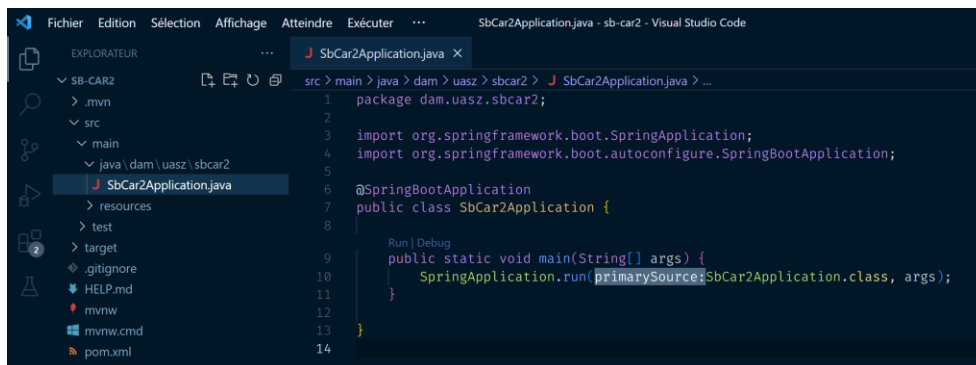
Lab_3 : JPA pour créer et accéder à une BD

Exercice 1

Créer une nouvelle application *Spring Boot* nommée **sb-car-2**, en prenant le soin d'ajouter les dépendances pour l'ORM JPA et la BD runtime H2.

spring init --dependencies=web,devtools,jpa,h2 --build=maven sb-car-2

Ouvrir le code dans vs code et vérifier que les dépendances sont bien présentes dans le fichier **pom.xml** et que l'on a l'arborescence ci-dessous :



I. Les concepts de bases : **ORM, JPA, et Hibernate**

- **ORM** : Object-Relational Mapping ou « *Mapping Objet-Relationnel* » permet de faire le lien entre les objets de votre application et les données stockées dans une base de données relationnelle. Il permet donc de travailler avec des données en utilisant des objets et des méthodes plutôt que des requêtes **SQL**.
- **Java Persistence API (JPA)** est un ORM et fournit une correspondance objet-relationnel pour les développeurs Java.
- **Hibernate** est l'implémentation JPA basée sur Java la plus populaire et est utilisée par défaut dans Spring Boot. Hibernate est un produit mature et est largement utilisé dans des applications à grande échelle.

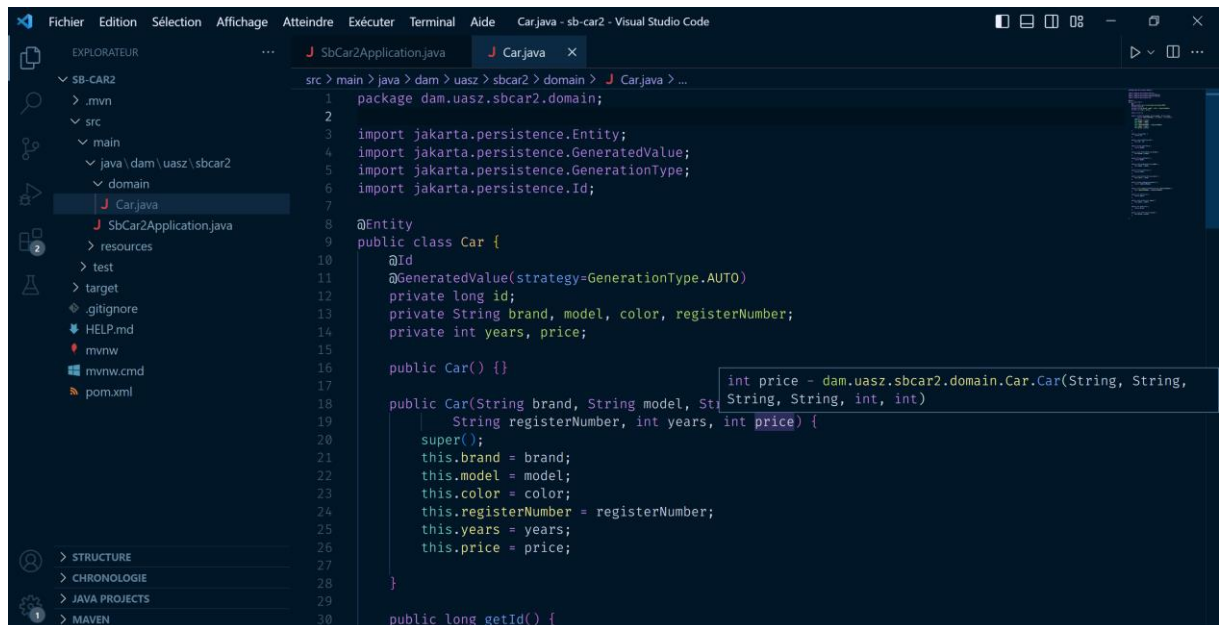
II. Création des classes d'entité

En **Java Persistence API (JPA)**, une classe d'entité est une simple classe Java qui représente une table de base de données et est annotée avec l'annotation **@Entity**. Une classe d'entité doit suivre les conventions de nommage **JavaBean** et avoir des champs privés avec des méthodes getter et setter publiques.

Lorsque l'application est initialisée, **JPA** crée une table correspondante dans la base de données avec le même nom que la classe d'entité. Cependant, si vous souhaitez utiliser un nom différent pour la table dans la base de données, vous pouvez utiliser l'annotation **@Table** dans votre classe d'entité et spécifier le nom souhaité comme valeur pour l'annotation.

Une classe d'entité peut également avoir d'autres annotations pour fournir des informations supplémentaires sur l'entité, telles que l'annotation **@Id** pour indiquer le champ de clé primaire, **@Column** pour spécifier le nom et les attributs de la colonne, et **@GeneratedValue** pour indiquer comment la valeur de clé primaire est générée.

Nous allons créer une classe d'entité nommée Car dans le package **dam.uaszbcarz.domain**.
On aura l'arborescence ci-dessous :



Faudra ajouter tous les getters et setters. Puis on lance l'application pour constater que notre application s'est connectée à une BD (regarder aux lignes où on a HiakiPool-1) dont l'url est `url=jdbc:h2:mem:25bae819-1912-415b-a5b3-941b8d2bc99a user=SA`.

Nous n'avons écrit aucun code pour créer une BD et créer l'utilisateur associé, mais spring boot le fait pour nous grâce notamment à l'auto configuration et le principe du « convention over configuration ».



Il nous est possible également de modifier ces données par défaut en agissant sur le fichier **application.properties** dans le dossier **resource** en mettant notamment les information ci-dessous.

```

application.properties x Car.java
1 spring.datasource.url=jdbc:h2:mem:testdb
2 spring.jpa.show-sql=true
3 spring.datasource.username=flow
4

```

On relance l'application pour constater que les noms sont devenus maintenant plus compréhensibles. La ligne 2 ajoutée dans le fichier ci-dessus permet d'afficher les requêtes SQL exécutées.

```

MINGW64/C:/codes/java/sb-car2
[INFO] Attaching agents: []

Spring Boot :: (v3.8.6)

2023-04-25T07:34:32.005Z INFO 16112 --- [ restartedMain] dan.uszr.sbcar2.SbCar2Application : Starting SbCar2Application using Java 17.0.6 with PID 16112 (C:/codes/java/sb-car2/target/classes started by DELL in C:/codes/java/sb-car2)
2023-04-25T07:34:32.011Z INFO 16112 --- [ restartedMain] dan.uszr.sbcar2.SbCar2Application : No active profile set, falling back to 1 default profile: 'default'
2023-04-25T07:34:32.122Z INFO 16112 --- [ restartedMain] o.devtools.properties.DefaultPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
2023-04-25T07:34:32.122Z INFO 16112 --- [ restartedMain] o.devtools.properties.DefaultPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2023-04-25T07:34:32.925Z INFO 16112 --- [ restartedMain] s.d.f.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2023-04-25T07:34:32.949Z INFO 16112 --- [ restartedMain] s.d.f.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 15 ms. Found 0 JPA repository interfaces.
2023-04-25T07:34:33.770Z INFO 16112 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-04-25T07:34:33.783Z INFO 16112 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-04-25T07:34:33.783Z INFO 16112 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.8]
2023-04-25T07:34:33.874Z INFO 16112 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-04-25T07:34:33.876Z INFO 16112 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1749 ms
2023-04-25T07:34:33.915Z INFO 16112 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-04-25T07:34:34.142Z INFO 16112 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2023-04-25T07:34:34.142Z INFO 16112 --- [ restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:testdb'
2023-04-25T07:34:34.325Z INFO 16112 --- [ restartedMain] o.hibernate.jpa.internal.util.LogHelper : HHH000284: Processing PersistenceUnitInfo [name: default]
2023-04-25T07:34:34.408Z INFO 16112 --- [ restartedMain] org.hibernate.Version : HHH000412: Hibernate ORM core version 6.1.7.Final
2023-04-25T07:34:34.756Z INFO 16112 --- [ restartedMain] SQL dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
Hibernate: drop table if exists car cascade
Hibernate: drop sequence if exists car_seq
Hibernate: create sequence car_seq start with 1 increment by 50
Hibernate: create table car (id bigint not null, brand varchar(255), color varchar(255), model varchar(255), price integer not null, register_number varchar(255), years integer not null, primary key (id))
2023-04-25T07:34:35.678Z INFO 16112 --- [ restartedMain] o.h.e.t.j.p.l.2taPlatformInitiator : HHH000400: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2023-04-25T07:34:35.692Z INFO 16112 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2023-04-25T07:34:35.758Z WARN 16112 --- [ restartedMain] jpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2023-04-25T07:34:36.158Z INFO 16112 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2023-04-25T07:34:36.191Z INFO 16112 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-04-25T07:34:36.208Z INFO 16112 --- [ restartedMain] dan.uszr.sbcar2.SbCar2Application : Started SbCar2Application in 4.667 seconds (process running for 5.23)

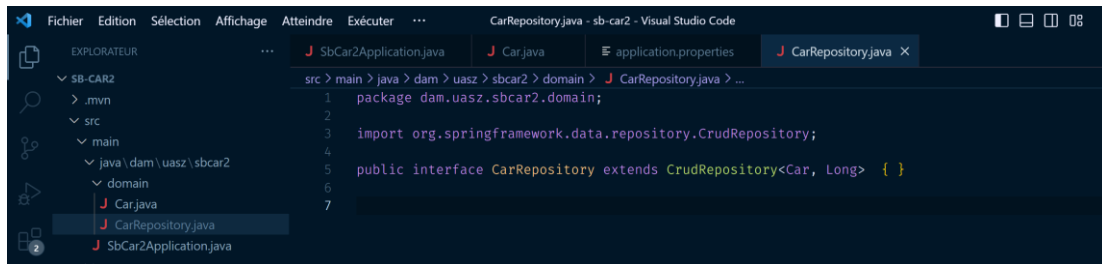
```

La BD est accessible à l'url <http://localhost:8080/h2-console>. Nous constatons que notre table est bien présente. On peut également spécifier le mot de passe dans **application.properties** comme on a fait pour le « *username* ».

III. Creation des repositories CRUD

Spring Boot Data JPA fournit une interface **CrudRepository** pour les opérations CRUD. Elle fournit des fonctionnalités CRUD à notre classe d'entité.

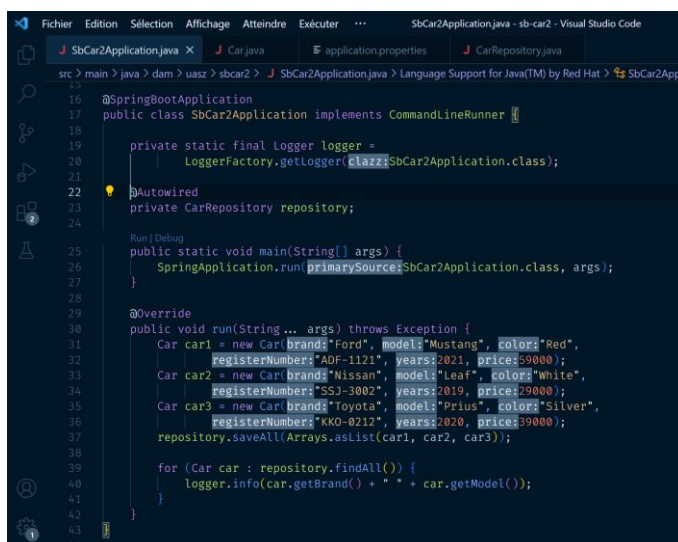
Nous allons créer un nouveau fichier dans le package **domain**. Il s'agit d'une interface **CarRepository** qui étend **CrudRepository**. Pour le moment nous n'ajouterons pas de nouvelles méthodes mais on peut faire des opérations CRUD de base.



Les arguments de type **<Car, Long>** définissent que ceci est le répertoire pour la classe d'entité Car et que le type du champ ID est Long. L'interface **CrudRepository** fournit plusieurs méthodes CRUD que nous pouvons maintenant commencer à utiliser. Le tableau suivant répertorie les méthodes les plus couramment utilisées.

Method	Description
long count()	Returns the number of entities
Iterable<T> findAll()	Returns all items of a given type
Optional<T> findById(ID id)	Returns one item by ID
void delete(T entity)	Deletes an entity
void deleteAll()	Deletes all the entities in the repository
<S extends T> save(S entity)	Saves an entity
List<S> saveAll(Iterable<S> entities)	Saves multiple entities

Maintenant, on peut ajouter des données de départ à notre base de données runtime H2. Pour cela, nous utiliserons l'interface **CommandLineRunner** de Spring Boot. L'interface **CommandLineRunner** nous permet d'exécuter du code supplémentaire avant que l'application ne soit entièrement démarrée. Par conséquent, c'est un bon point pour ajouter des données de démonstration à votre base de données. La classe principale de votre application Spring Boot implémente l'interface **CommandLineRunner**. Le code de notre classe principale ressemblera à ceci :

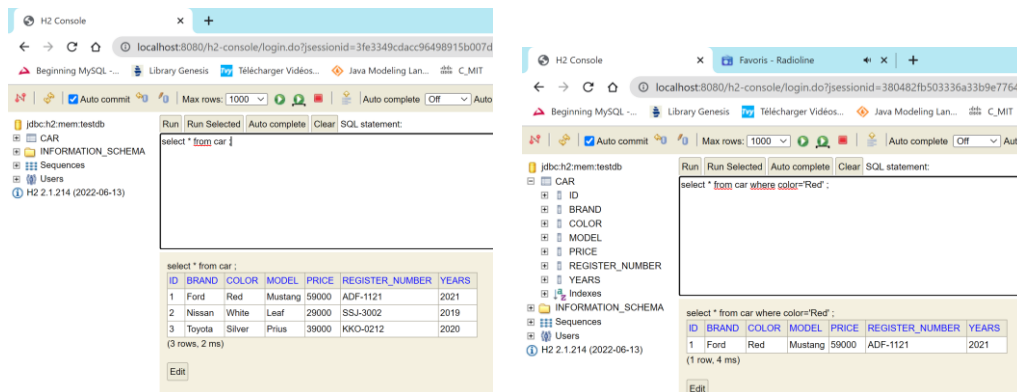


Nous pouvons constater que les requêtes sql d'insertion sont affichées dans la console. Nous avons un **logger** pour voir les trafics dans la console. A noter que l'on peut choisir les types de log à afficher dans le fichier **application.properties** en ajoutant la ligne

logging.level.root=INFO. La valeur par défaut c'est INFO, la valeur mais on peut mettre ERREUR, WARN ou DEBUG, ce dernier donne plus de détails.

```
MINGW64/C:/codes/java/sb-car2
2023-04-25T08:19:31.795Z INFO 15972 --- [ restartedMain ] o.s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 53 ms. Found 3 JPA repository interfaces.
2023-04-25T08:19:32.882Z INFO 15972 --- [ restartedMain ] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-04-25T08:19:32.901Z INFO 15972 --- [ restartedMain ] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-04-25T08:19:32.901Z INFO 15972 --- [ restartedMain ] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.0]
2023-04-25T08:19:33.009Z INFO 15972 --- [ restartedMain ] o.a.c.c.c.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-04-25T08:19:33.012Z INFO 15972 --- [ restartedMain ] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2876 ms
2023-04-25T08:19:33.061Z INFO 15972 --- [ restartedMain ] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-04-25T08:19:33.315Z INFO 15972 --- [ restartedMain ] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection conn0: url=jdbc:h2:mem:testdb user=FLOW
2023-04-25T08:19:33.319Z INFO 15972 --- [ restartedMain ] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2023-04-25T08:19:33.337Z INFO 15972 --- [ restartedMain ] o.s.i.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:testdb'
2023-04-25T08:19:33.528Z INFO 15972 --- [ restartedMain ] o.hibernate.jpa.internal.util.LogHelper : HHH000294: Processing PersistenceUnitInfo [name: default]
2023-04-25T08:19:33.588Z INFO 15972 --- [ restartedMain ] org.hibernate.Version : HHH000012: Hibernate ORM core version 6.1.7.Final
2023-04-25T08:19:33.938Z INFO 15972 --- [ restartedMain ] SQL dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
Hibernate: drop table if exists car cascade
Hibernate: drop sequence if exists car_seq
Hibernate: create sequence car_seq start with 1 increment by 50
Hibernate: create table car (id bigint not null, brand varchar(255), color varchar(255), model varchar(255), price integer not null, register_number varchar(255), years integer not null, primary key (id))
2023-04-25T08:19:34.797Z INFO 15972 --- [ restartedMain ] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.LocalContainerEntityManagerFactoryBean]
2023-04-25T08:19:34.808Z INFO 15972 --- [ restartedMain ] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2023-04-25T08:19:35.166Z WARN 15972 --- [ restartedMain ] jpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering.
2023-04-25T08:19:35.478Z INFO 15972 --- [ restartedMain ] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2023-04-25T08:19:35.519Z INFO 15972 --- [ restartedMain ] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-04-25T08:19:35.532Z INFO 15972 --- [ restartedMain ] dan.uasz.sbcars2.SBCar2Application : Started SBCar2Application in 5.217 seconds (process running for 5.764)
Hibernate: select next value for car_seq
Hibernate: select next value for car_seq
Hibernate: insert into car (brand, color, model, price, register_number, years, id) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into car (brand, color, model, price, register_number, years, id) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into car (brand, color, model, price, register_number, years, id) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: select c1_0.id,c1_0.brand,c1_0.color,c1_0.model,c1_0.price,c1_0.register_number,c1_0.years from car c1_0
2023-04-25T08:19:35.773Z INFO 15972 --- [ restartedMain ] dan.uasz.sbcars2.SBCar2Application : Ford Mustang
2023-04-25T08:19:35.773Z INFO 15972 --- [ restartedMain ] dan.uasz.sbcars2.SBCar2Application : Nissan Leaf
2023-04-25T08:19:35.774Z INFO 15972 --- [ restartedMain ] dan.uasz.sbcars2.SBCar2Application : Toyota Prius
```

On peut ouvrir notre interface dans H2 afin de constater si nos données sont insérées.



Il est également possible d'ajouter ses propres requêtes dans le « repository »¹.

```
3 import java.util.List;
4
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.CrudRepository;
7 import org.springframework.data.repository.query.Param;
8
9 public interface CarRepository extends CrudRepository<Car, Long> {
10     // Fetch cars by brand
11     List<Car> findByBrand(@Param("brand") String brand);
12
13     // Fetch cars by color
14     List<Car> findByColor(@Param("color") String color);
15
16     // Fetch cars by brand and model
17     List<Car> findByBrandAndModel(String brand, String model);
18
19     // Fetch cars by brand or color
20     List<Car> findByBrandOrColor(String brand, String color);
21
22     // Fetch cars by model using SQL
23     @Query("select c from Car c where c.model = ?1")
24     List<Car> findByModel(String model);
25 }
26
27
```

¹ L'intérêt de cette partie dans le lab4

IV. Ajout de relations entre tables

Considérons ce modèle ci-dessous :

- Une voiture a un seul propriétaire qui peut avoir plusieurs voitures.

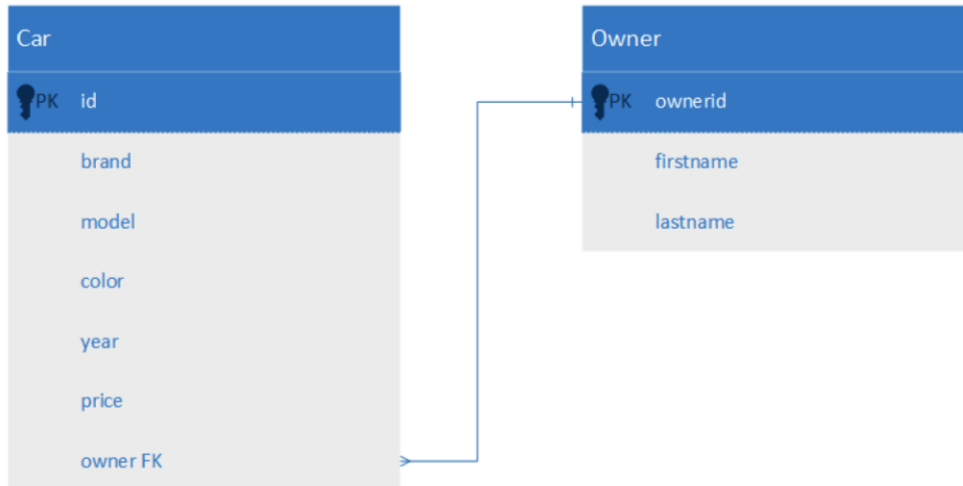


Figure 3.12 – OneToMany relationship

Créer la classe d'entité **Owner** et son **repository** correspondant dans le package **domain**.

```
> Car.java
> CarRepository.java
> Owner.java
> OwnerRepository.java
> src/main/resources
> src/test/java
```

Le code de la classe **Car** sera revue comme suit en ajoutant la colonne **owner** : la clé étrangère. Le mot clé **year** est un mot clé dans sql, peut donc engendrer quelques erreurs. Il faut ajouter des getters et setters à la classe **Car** pour le nouvel attribut **owner**.

```
Fichier Edition Sélection Affichage Atteindre Exécuter Terminal Aide Car.java - sb-car2 - Visual Studio Code
src > main > java > dam > uasz > sbcar2 > domain > Car.java > Car > Car(String, String, String, String, int, int, Owner)

12 @Entity
13 public class Car {
14     @Id
15     @GeneratedValue(strategy=GenerationType.AUTO)
16     private long id;
17     private String brand, model, color, registerNumber;
18
19     @Column(name="year")
20     private int year;
21
22     private int price;
23
24     public Car() {}
25
26     public Car(String brand, String model, String color,
27                String registerNumber, int year, int price, Owner owner) {
28         super();
29         this.brand = brand;
30         this.model = model;
31         this.color = color;
32         this.registerNumber = registerNumber;
33         this.year = year;
34         this.price = price;
35         this.owner = owner;
36
37
38     @ManyToOne(fetch = FetchType.LAZY)
39     @JoinColumn(name = "owner")
40     private Owner owner;
41 }
```

La relation un-à-plusieurs peut être ajoutée en utilisant les annotations **@ManyToOne** et **@OneToMany**. Dans la classe d'entité **Car**, qui contient une clé étrangère, vous devez définir

la relation avec l'annotation **@ManyToOne**. Vous devriez également ajouter les accesseurs pour le champ **owner**. Il est recommandé d'utiliser **FetchType.LAZY** pour toutes les associations. Pour les relations de type **"toMany"**, c'est la valeur par défaut, mais pour les relations de type **"toOne"**, vous devriez la définir. **FetchType** définit la stratégie de récupération des données à partir de la base de données. La valeur peut être soit **EAGER** soit **LAZY**. Dans notre cas, la stratégie **"lazy"** signifie que lorsque le propriétaire est récupéré de la base de données, toutes les voitures associées au propriétaire seront récupérées lorsque cela est nécessaire. **"Eager"** signifie que les voitures seront récupérées immédiatement par le propriétaire.

Le code de la classe **Owner** est présentée ci-dessous. Faudra ajouter les getters et setters pour **firstname** et **lastname**. Dans l'entité **Car**, la relation est définie avec l'annotation **@OneToMany**. Le type de champ est **List<Car>** car le propriétaire peut avoir plusieurs voitures.

```

7 import jakarta.persistence.Id;
8 import jakarta.persistence.OneToMany;
9 import java.util.List;
10 import jakarta.persistence.CascadeType;
11
12 @Entity
13 public class Owner {
14     @Id
15     @GeneratedValue(strategy=GenerationType.AUTO)
16     private long ownerid;
17     private String firstname, lastname;
18
19     public Owner() {}
20
21     public Owner(String firstname, String lastname) {
22         super();
23         this.firstname = firstname;
24         this.lastname = lastname;
25     }
26
27     @OneToMany(cascade=CascadeType.ALL, mappedBy="owner")
28     private List<Car> cars;
29
30     public List<Car> getCars() {
31         return cars;
32     }
33
34     public void setCars(List<Car> cars) {
35         this.cars = cars;
36     }

```

L'annotation **@OneToMany** a deux attributs que nous utilisons. L'attribut **cascade** définit comment la cascade affecte les entités en cas de suppressions ou de mises à jour. Le paramètre **ALL** signifie que toutes les opérations sont en cascade. Par exemple, si le propriétaire est supprimé, les voitures qui sont liées à ce propriétaire sont également supprimées. Le paramètre **mappedBy="owner"** nous indique que la classe **Car** a le champ **owner**, qui est la clé étrangère pour cette relation.

select * from car;

ID	BRAND	COLOR	MODEL	PRICE	REGISTER_NUMBER	year	OWNER
1	Ford	Red	Mustang	59000	ADF-1121	2021	1
2	Nissan	White	Leaf	29000	SSJ-3002	2019	2
3	Toyota	Silver	Prius	39000	KKO-0212	2020	2

(3 rows, 4 ms)

select * from owner;

OWNERID	FIRSTNAME	LASTNAME
1	John	Johnson
2	Mary	Robinson

(2 rows, 2 ms)

Le code de la classe principale est donné ci-dessous :

```

1 package com.dam.uas2.sbcar2;
2
3 import java.util.Arrays;
4
5 import org.slf4j.Logger;
6 import org.slf4j.LoggerFactory;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.boot.CommandLineRunner;
9 import org.springframework.boot.SpringApplication;
10 import org.springframework.boot.autoconfigure.SpringBootApplication;
11
12 import com.dam.uas2.sbcar2.domain.*;
13
14 @SpringBootApplication
15 public class SbCar2Application implements CommandLineRunner {
16
17     private static final Logger logger =
18         LoggerFactory.getLogger(SbCar2Application.class);
19
20     @Autowired
21     private CarRepository repository;
22
23
24     @Autowired
25     private OwnerRepository orepository;
26
27
28     public static void main(String[] args) {
29         SpringApplication.run(SbCar2Application.class, args);
30     }
31
32     @Override
33     public void run(String... args) throws Exception {
34         // Add owner objects and save these to db
35         Owner owner1 = new Owner("John", "Johnson");
36         Owner owner2 = new Owner("Mary", "Robinson");
37         orepository.saveAll(Arrays.asList(owner1, owner2));
38
39         // Add car object and link to owners and save these to db
40         Car car1 = new Car("Ford", "Mustang", "Red",
41             "ADF-1121", 2021, 59000, owner1);
42         Car car2 = new Car("Nissan", "Leaf", "White",
43             "SSJ-3002", 2019, 29000, owner2);
44         Car car3 = new Car("Toyota", "Prius", "Silver",
45             "KKO-0212", 2020, 39000, owner2);
46         repository.saveAll(Arrays.asList(car1, car2, car3));
47
48         for (Car car : repository.findAll()) {
49             logger.info(car.getBrand() + " " + car.getModel());
50         }
51     }
52 }

```

V. Utilisation d'une BD relationnelle

Pour la suite, nous allons utiliser une BD **MariaDB** à la place de notre BD **in-memory H2**. **MariaDB** est une base de données relationnelle open source largement utilisée. Il faudra l'installer dans la suite de ce cours. Il peut venir avec un serveur comme *xampp* ou vous pouvez le télécharger et l'installer.

Les tables de la base de données sont toujours créées automatiquement par **JPA**. Cependant, avant de lancer notre application, nous devons créer une base de données pour celle-ci. Un autre SGBD relationnelle fera l'affaire.

```

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database cardb ;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]>

```

Dans Spring Boot, ajoutez une dépendance cliente Java **MariaDB** au fichier **pom.xml** et supprimez la dépendance **H2** car nous n'en avons plus besoin :

```

<dependency>
  <groupId>org.mariadb.jdbc</groupId>
  <artifactId>mariadb-java-client</artifactId>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.36</version>
</dependency>
</dependencies>

```

Ajouter ces infos de config dans **application.properties** :


```

1 spring.datasource.url=jdbc:mariadb://localhost:3306/cardb
2 spring.datasource.username=root
3 spring.datasource.password=
4 spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
5 spring.jpa.generate-ddl=true
6 spring.jpa.hibernate.ddl-auto=create-drop
7

```

Le paramètre **spring.jpa.generate-ddl** définit si **JPA** doit initialiser la base de données (*true/false*). Les valeurs possibles sont *none*, *validate*, *update*, *create* et *create-drop*. La valeur par défaut dépend de votre base de données. Si vous utilisez une base de données intégrée comme **H2**, la valeur par défaut est *create-drop*, sinon, la valeur par défaut est *none*. *create-drop* signifie que la base de données est créée lorsque l'application démarre et supprimée lorsque l'application s'arrête. La valeur *create* ne crée la base de données que lorsque l'application démarre. La valeur *update* crée la base de données et met à jour le schéma si celui-ci a changé.

Si on redémarre le serveur, nous voyons que la BD est peuplée par les données initiales.

```

MariaDB [cardb]> show tables ;
+-----+
| Tables_in_cardb |
+-----+
| car              |
| car_seq          |
| owner            |
| owner_seq        |
+-----+
4 rows in set (0.001 sec)

MariaDB [cardb]> select * from car ;
+-----+
| id | brand | color | model | price | register_number | year | owner |
+-----+
| 1  | Ford  | Red   | Mustang | 59000 | ADF-1121       | 2021 | 1  |
| 2  | Nissan | White | Leaf   | 29000 | SSJ-3002       | 2019 | 2  |
| 3  | Toyota | Silver | Prius  | 39000 | KK0-0212       | 2020 | 2  |
+-----+
3 rows in set (0.001 sec)

MariaDB [cardb]>

```

Exercice 2

NB : Pour faire cet exercice, il vous est demandé de cloner l'exercice précédent. Le code de l'exercice 1 sera utilisé dans le prochain lab.

On considère maintenant qu'une voiture peut avoir plusieurs propriétaires. Essayer d'adapter les codes pour gérer les relations de plusieurs à plusieurs.

Ci-dessous, on constate qu'une table supplémentaire est automatiquement ajoutée.

La table est appelée **car_owner**.

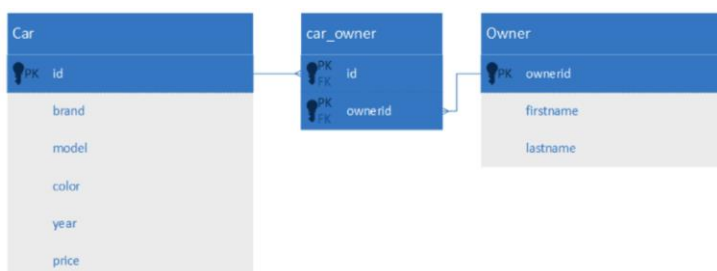


Figure 3.18 – ManyToMany relationship

La table d'association est un type particulier de table qui gère la relation de plusieurs à plusieurs entre deux tables. La table d'association est définie en utilisant l'annotation **@JoinTable**. Avec cette annotation, nous pouvons définir le nom de la table de liaison et les colonnes de liaison.

Hint 🧐

Les classes entités sont modifiées comme suit :

```

Car.java x CarRepository.java Owner.java OwnerRepository.java SbCarApplication.java
27 private int price;
28
29 public Car() {}
30
31 public Car(String brand, String model, String color,
32 String registerNumber, int year, int price, Set<Owner> owners) {
33 super();
34 this.brand = brand;
35 this.model = model;
36 this.color = color;
37 this.registerNumber = registerNumber;
38 this.year = year;
39 this.price = price;
40 this.owners = owners;
41 }
42
43 @ManyToMany(mappedBy="cars")
44 private Set<Owner> owners = new HashSet<Owner>();
45 public Set<Owner> getOwners() {
46 return owners;
47 }
48
49 public void setOwner(Set<Owner> owners) {
50 this.owners = owners;
51 }
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

sb-car-1 - SbCarApplication [Spring Boot App] C:\Users\DELL\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.6.v20230204-1729\jre\bin\javaw.exe (25 avr. 2023, 23:45:30) [p
Hibernate: drop sequence car_seq
Hibernate: create sequence car_seq start with 1 increment by 50
Hibernate: create sequence owner_seq start with 1 increment by 50
Hibernate: create table car (id bigint not null, brand varchar(255), color varchar(255), model varchar(255), price integer not null,
Hibernate: create table car_owner (ownerid bigint not null, id bigint not null, primary key (ownerid, id))
Hibernate: create table owner (ownerid bigint not null, firstname varchar(255), lastname varchar(255), primary key (ownerid))
Hibernate: alter table if exists car_owner add constraint FKofv7bce0mwfufud5r1k21w3p foreign key (ownerid) references owner
Hibernate: alter table if exists car_owner add constraint FKrd5f5fk3bil77mf9m6vqixx foreign key (id) references owner
2023-04-25T23:45:37.2752 INFO 12572 --- [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform im
2023-04-25T23:45:37.2902 INFO 12572 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFa
2023-04-25T23:45:37.9912 INFO 12572 --- [ restartedMain] jpaBaseConfiguration$JpaWebConfiguration : Spring.jpa.open-in-view is enab
2023-04-25T23:45:38.3872 INFO 12572 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on
2023-04-25T23:45:38.4412 INFO 12572 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080
2023-04-25T23:45:38.4592 INFO 12572 --- [ restartedMain] com.dam.uasz.sbcar1.SbCar1Application : Started SbCar1Application in 5.
Hibernate: select next value for owner_seq
Hibernate: select next value for owner_seq
Hibernate: insert into owner (firstname, lastname, ownerid) values (?, ?, ?)
Hibernate: insert into owner (firstname, lastname, ownerid) values (?, ?, ?)
Hibernate: select next value for car_seq
Hibernate: select next value for car_seq
Hibernate: insert into car (brand, color, model, price, register_number, "year", id) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into car (brand, color, model, price, register_number, "year", id) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into car (brand, color, model, price, register_number, "year", id) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: select c1.id,c1.brand,c1.color,c1.model,c1.price,c1.register_number,c1."year" from car c1

```

H2 Console

localhost:8080/h2-console/login.do?sessionId=42293aa552632a43

Beginning MySQL ... Library Genesis Télécharger Videos... Java Modeling Lan...

Auto commit Max rows: 1000 Auto complete Off

jdbc:h2:mem:testdb

Run Run Selected Auto complete Clear SQL statement:

select * from owner

select * from owner:

OWNERID	FIRSTNAME	LASTNAME
1	John	Johnson
2	Mary	Robinson

(2 rows, 1 ms)

Edit

Finir l'exercice2 😞