

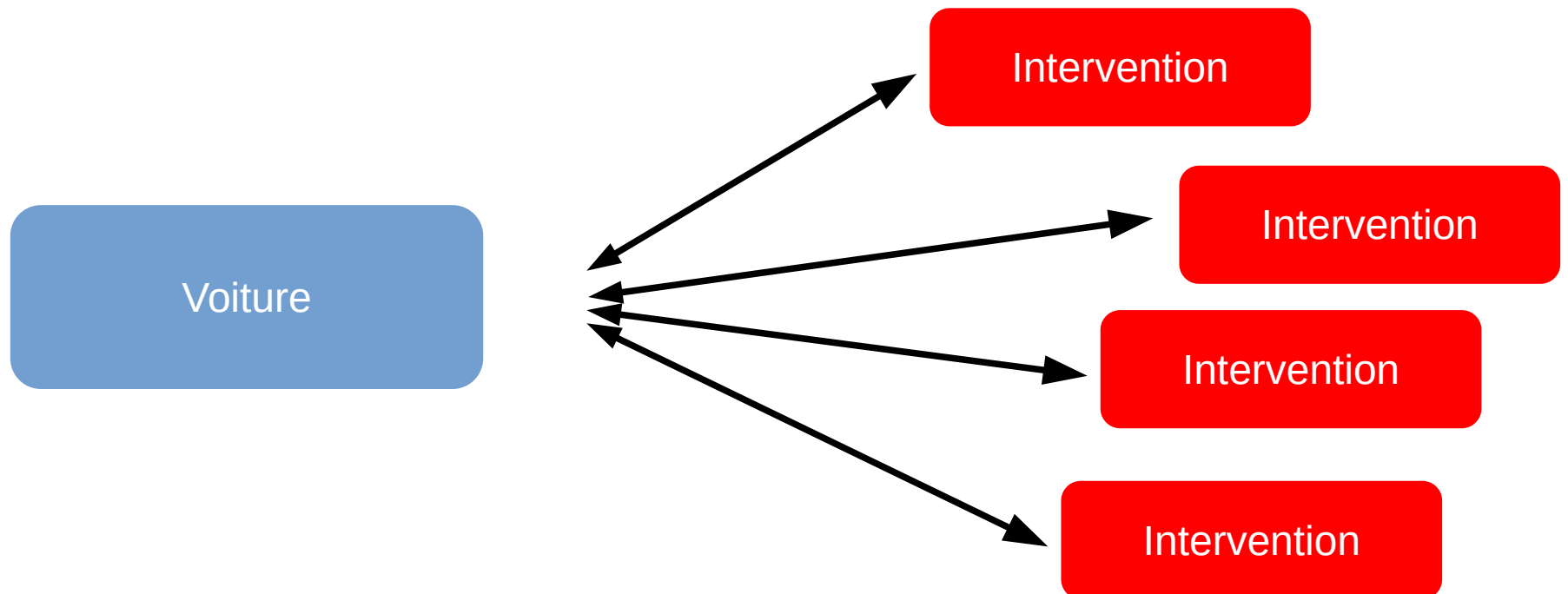


# Jpa - relations

@ManyToOne

# @OneToMany bi directionnel

## @ManyToOne



# Notre règle métier: Cascading nb une spécificité sur le delete operation

- Si on supprime une intervention on ne supprimera pas la voiture
- Si on supprime la voiture , on ne supprimera pas les interventions

Notre programme :

préparer le travail de dev – définir les tables

créer a classe intervention

mettre à jour la classe voiture

développer une classe exécutable pour manipuler les objets

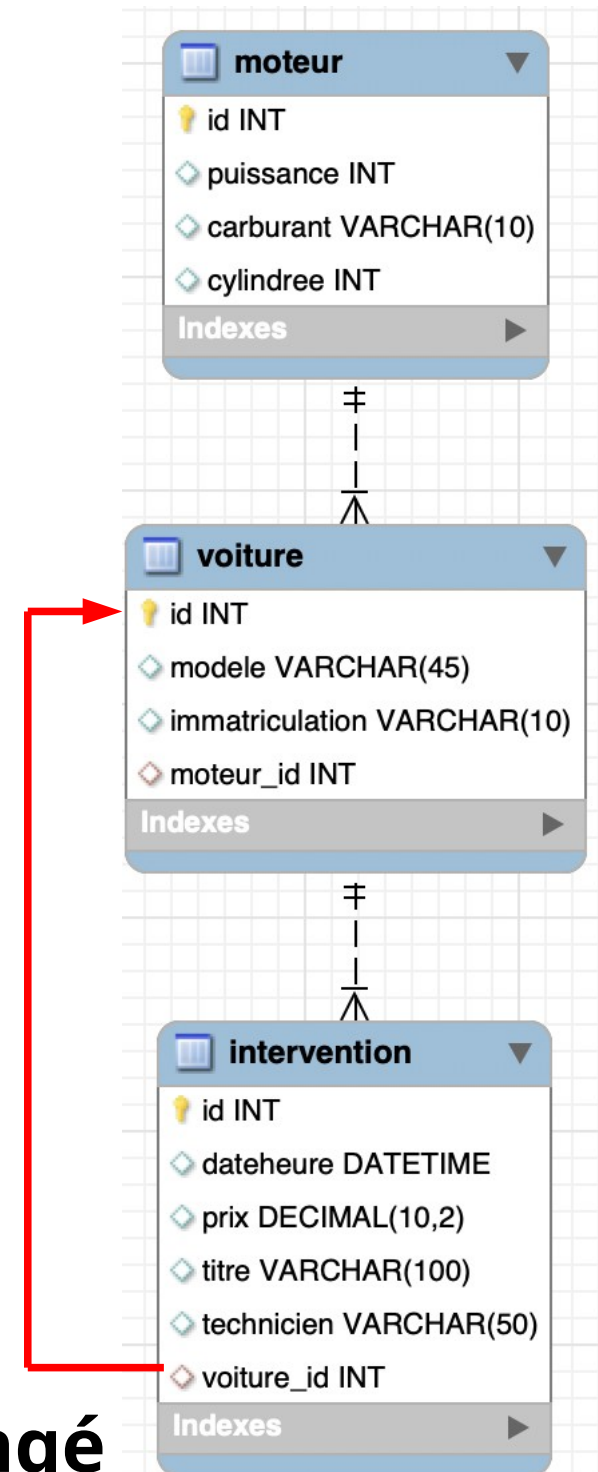
# Code sql pour intervention

```
-- Table structure for table `intervention`  
DROP TABLE IF EXISTS `intervention`;  
CREATE TABLE `intervention` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `dateheure` DATETIME DEFAULT NULL,  
  `prix` DECIMAL( 10, 2 ) DEFAULT NULL,  
  `titre` varchar (100) DEFAULT NULL,  
  `technicien` varchar (50) DEFAULT NULL,  
  `voiture_id` INT DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `FK_VOITURE_idx` (`voiture_id`),  
  CONSTRAINT `FK_to_voiture_id`  
  FOREIGN KEY (`voiture_id`)  
  REFERENCES `voiture` (`id`)  
  
  ON DELETE NO ACTION ON UPDATE NO ACTION  
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT  
CHARSET=latin1;
```

Le code sql pour voiture reste inchangé

6 oct. 2021

CJD-Formation



# Coder la classe Intervention.java qui "mappera" (matchera, correspondra) avec cette table

```
@Entity
@Table(name="intervention")
public class Intervention {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column
    private int id ;

    @Column
    private LocalDateTime dateheure;

    @Column
    private Double prix;

    @Column
    private String titre;

    @Column
    private String technicien;

    @ManyToOne(cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                        CascadeType.DETACH, CascadeType.REFRESH})
    @JoinColumn(name="voiture_id")
    private Voiture voiture;

    // getters & setters . . .
    // Constructors . . .
}
```

intervention	
id	INT
dateheure	DATETIME
prix	DECIMAL(10,2)
titre	VARCHAR(100)
technicien	VARCHAR(50)
voiture_id	INT
Indexes	

Ne pas cascader la suppression

# Maj de la classe voiture

```
@Entity
@Table(name="voiture")
public class Voiture {
```

```
...
```

```
@OneToOne(cascade=CascadeType.ALL)
@JoinColumn(name="moteur_id")
private Moteur moteur;
```

```
@OneToMany(mappedBy="voiture",
            cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                      CascadeType.DETACH, CascadeType.REFRESH})
private List<Intervention> interventions;
```

```
//getters & setters
```

```
public List<Intervention> getInterventions() {
    return interventions;
}

public void setInterventions(List<Intervention> interventions) {
    this.interventions = interventions;
}
```

mappedBy cible le  
champ de la relation  
coté Interventions

```
public class Intervention {
    ...

    @ManyToOne
    @JoinColumn(name="voiture_id")
    private Voiture voiture;
```

Ne pas cascader  
la suppression

# Une méthode pratique pour les relations bi directionnelles

```
@Entity
@Table(name="voiture")
public class Voiture {

    . . .

    @OneToMany(mappedBy="voiture" ,
                cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                           CascadeType.DETACH, CascadeType.REFRESH})
    private List<Intervention> interventions;

    // méthode pratique pour les relations bi directionnelles oneToMany
    public void add (Intervention intervention) {
        if(interventions == null ){
            interventions = new ArrayList<>();
        }

        interventions.add(intervention);
        intervention.setVoiture(this);
    }
}
```

Lien  
Bi-directionnel

# Une méthode pratique pour les relations bi directionnelles

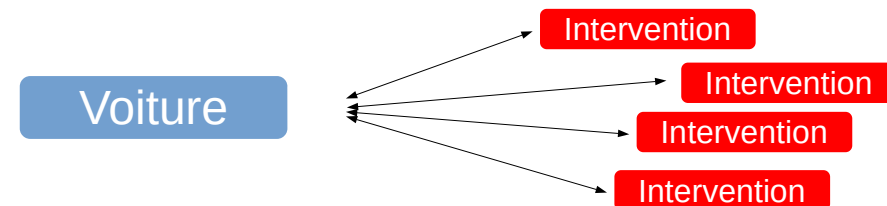
```
@Entity
@Table(name="voiture")
public class Voiture {

    . . .

    @OneToMany(mappedBy="voiture" ,
                cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                           CascadeType.DETACH, CascadeType.REFRESH})
    private List<Intervention> interventions;

    // méthode pratique pour les relations bi directionnelles oneToMany
    public void add (Intervention intervention) {
        if(interventions == null ){
            interventions = new ArrayList<>();
        }

        interventions.add(intervention);
        intervention.setVoiture(this);
    }
}
```





# Créer la classe exécutable

Ce que nous voulons faire pour montrer comment fonctionne cette relation, c'est récupérer une Voiture dans la base et lui ajouter des Interventions

```
public class CreateVoitureDemo {  
  
    public static void main(String[] args) {  
  
        // create session factory  
        SessionFactory factory = new Configuration()  
            .configure("hibernate.cfg.xml")  
            .addAnnotatedClass(Voiture.class)  
            .addAnnotatedClass(Moteur.class)  
            .addAnnotatedClass(Intervention.class)  
            .buildSessionFactory();  
  
        // déclarer session  
        Session session = null;  
        try {  
            // récupérer une session & ouvrir une transaction  
            session = factory.getCurrentSession();  
            session.beginTransaction();  
        }  
    }  
}
```

```
public class CreateVoitureDemo {

    public static void main(String[] args) {

        . . .

        // récupérer une session & ouvrir une transaction
        session = factory.getCurrentSession();
        session.beginTransaction();

        // créer des objets
        Voiture v= new Voiture("clio", "444-ddd-44");
        Moteur m = new Moteur (115, "diesel",1400);

        //associer les objets
        v.setMoteur(m);

        // récupérer une session & ouvrir une transaction
        session = factory.getCurrentSession();
        session.beginTransaction();

        // sauvegarder la voiture
        System.out.println("voiture : "+v);
        session.save(v);

        //commit transaction
        session.getTransaction().commit();

        System.out.println("Terminé !");

        . . .
    }
}
```

# On peut à présent exécuter cette classe

```
août 03, 2020 3:14:57 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
voiture : Voiture [id=null, modele=clio, immat=444-ddd-44]
Hibernate: insert into Moteur (carburant, cylindree, puissance) values (?, ?, ?)
Hibernate: insert into voiture (immatriculation, modele, moteur_id) values (?, ?, ?)
Terminé !
août 03, 2020 3:14:58 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerCo
INFO: HHH10001002: Cleaning up connection pool [idbmysql://localhost:3306/ddd_05_000]
```


	id	modele	immatriculati...	moteur_id
▶	1	clio	444-ddd-44	1
	NULL	NULL	NULL	NULL

	id	puissance	carburant	cylindree
	1	115	diesel	1400
	NULL	NULL	NULL	NULL

Maintenant que nous avons des données voiture et moteur on va s'intéresser à la relation OneToMany entre Voiture et les Interventions

# Dans une nouvelle classe exécutable SVP

```
public class CreateInterventionDemo {  
  
    public static void main(String[] args) {  
  
        // create session factory  
        SessionFactory factory = new Configuration()  
            .configure("hibernate.cfg.xml")  
            .addAnnotatedClass(Voiture.class)  
            .addAnnotatedClass(Moteur.class)  
            .addAnnotatedClass(Intervention.class)  
            .buildSessionFactory();  
  
        // déclarer session  
        Session session = null;  
        try {  
            // récupérer une session & ouvrir une transaction  
            session = factory.getCurrentSession();  
            session.beginTransaction();  
  
            //commit transaction  
            session.getTransaction().commit();  
  
            System.out.println("Terminé !");  
  
            . . .  
        }  
    }  
}
```



**Ce que nous voulons faire c'est**  
**lire une Voiture**  
**créer des interventions**  
**lui ajouter ces interventions**  
**sauvegarder ces interventions**

```
// récupérer une session & ouvrir une transaction
session = factory.getCurrentSession();
session.beginTransaction();

// lire une Voiture

// créer des interventions

// lui ajouter ces interventions

// sauvegarder ces interventions

//commit transaction
session.getTransaction().commit();
```

```
// récupérer une session & ouvrir une transaction
session = factory.getCurrentSession();
session.beginTransaction();

// lire une Voiture
Long id= 1L;
Voiture v = session.get(Voiture.class, id);

// créer des interventions
Intervention i1= new Intervention("Petite Révision", 80.5,"A.
Didonk",LocalDateTime.of(2020,Month.OCTOBER,10,10,00,00) );
Intervention i2= new Intervention("Changement des pneus avant",
20. , "M. Imome",LocalDateTime.of(2019,Month.DECEMBER,5,14,30,00) );
Intervention i3= new Intervention("Vidange", 80.5,"S.
Ouraille",LocalDateTime.of(2017,Month.JUNE,21,8,45,00) );

// ajouter ces Interventions à la voiture
v.add(i1);
v.add(i2);
v.add(i3);

// sauvegarder ces interventions
session.save(i1);
session.save(i2);
session.save(i3);


//commit transaction
session.getTransaction().commit();

System.out.println("Terminé !");
```

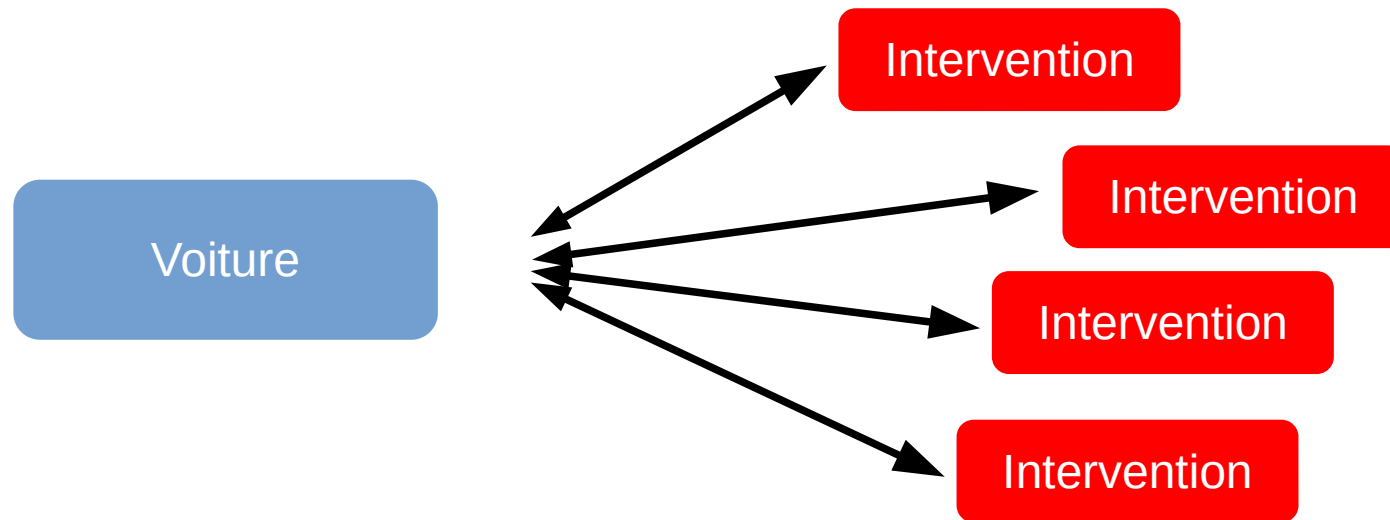
Executer

INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect  
Hibernate: select voiture0\_.id as id1\_2\_0\_, voiture0\_.immatriculation as immatric2\_2\_0\_, voiture0\_.modele as mode  
Hibernate: insert into intervention (dateheure, prix, technicien, titre, voiture\_id) values (?, ?, ?, ?, ?)  
Hibernate: insert into intervention (dateheure, prix, technicien, titre, voiture\_id) values (?, ?, ?, ?, ?)  
Hibernate: insert into intervention (dateheure, prix, technicien, titre, voiture\_id) values (?, ?, ?, ?, ?)  
Terminé !  
août 03, 2020 3:44:46 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl\$PoolL  
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/hdd\_05 one-to-many bidir?useSSL=false

Succès !

Result Grid							
Filter Rows: <input type="text" value="Search"/>							
Edit:    Export/Import:							
	id	dateheure	prix	titre	technicien	voiture_id	
▶	1	2020-10-10 08:00:00	80.50	Petite Révision	A. Didonk	1	
	2	2019-12-05 13:30:00	20.00	Changement des pneus avant	M. Imome	1	
	3	2017-06-21 06:45:00	80.50	Vidange	S. Ouraille	1	
	NULL	NULL	NULL	NULL	NULL	NULL	

**Nous souhaitons consulter les données des tables voiture moteur interventions**



**Je vous propose de créer une nouvelle classe executable pour vous montrer comment consulter une voiture et les Interventions qui lui sont rattachées.**



# Consulter une voiture et ses Interventions

```
public class GetVoitureInterventionsDemo{  
  
    public static void main(String[] args) {  
  
        // récupérer une session & ouvrir une transaction  
        session = factory.getCurrentSession();  
        session.beginTransaction();  
  
        //récupérer une voiture  
        Long id= 1L;  
        Voiture v = session.get(Voiture.class , id);  
  
        // afficher la voiture  
        System.out.println("voiture : "+v);  
  
        //afficher les Interventions associées  
        System.out.println("interventions : "+v.aetInterventions());  
  
        //commit transaction  
        session.getTransaction().commit();  
        System.out.println("Terminé !");  
    }  
}
```

Retournera  
une liste

Voiture

Intervention

Intervention

Intervention

Intervention

Executer

	id	dateheure	prix	titre	technicien	voiture_id
▶	1	2020-10-10 08:00:00	80.50	Petite Révision	A. Didonk	1
	2	2019-12-05 13:30:00	20.00	Changement des pneus avant	M. Imome	1
	3	2017-06-21 06:45:00	80.50	Vidange	S. Ouraille	1
	NULL	NULL	NULL	NULL	NULL	NULL

```

Markers Servers Properties Data Source Explorer Snippets Console
<terminated> GetVoitureInterventionsDemo [Java Application] /Library/Java/JavaVirtualMachines/openjdk-11.0.2.jdk/Contents/Home/bin/java (3 août 2020 à 16:03:16 – 16:03:18)
INFO: HHH10001005: using driver [com.mysql.cj.jdbc.Driver] at URL [jdbc:mysql://localhost:3306/bdd_05_one_to_many_bidir?useSSL=false&serverTimezone=UTC]
août 03, 2020 4:03:17 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {password=****, user=padawan}
août 03, 2020 4:03:17 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
août 03, 2020 4:03:17 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 1 (min=1)
août 03, 2020 4:03:17 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
Hibernate: select voiture0_.id as id1_2_0_, voiture0_.immatriculation as immatric2_2_0_, voiture0_.modele as modele3_2_0_, voiture0_.moteur_id as moteur_
voiture : Voiture [id=1, modele=clio, immat=444-ddd-44]
Hibernate: select interventi0_.voiture_id as voiture_6_0_0_, interventi0_.id as id1_0_0_, interventi0_.id as id1_0_1_, interventi0_.dateheure as dateheur
interventions : [Intervention [id=1, dateheure=2020-10-10T10:00, titre=Petite Révision, prix=80.5, technicien=A. Didonk], Intervention [id=2, dateheure=2
Terminé !
août 03, 2020 4:03:18 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PoolState stop
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/bdd_05_one_to_many_bidir?useSSL=false&serverTimezone=UTC]

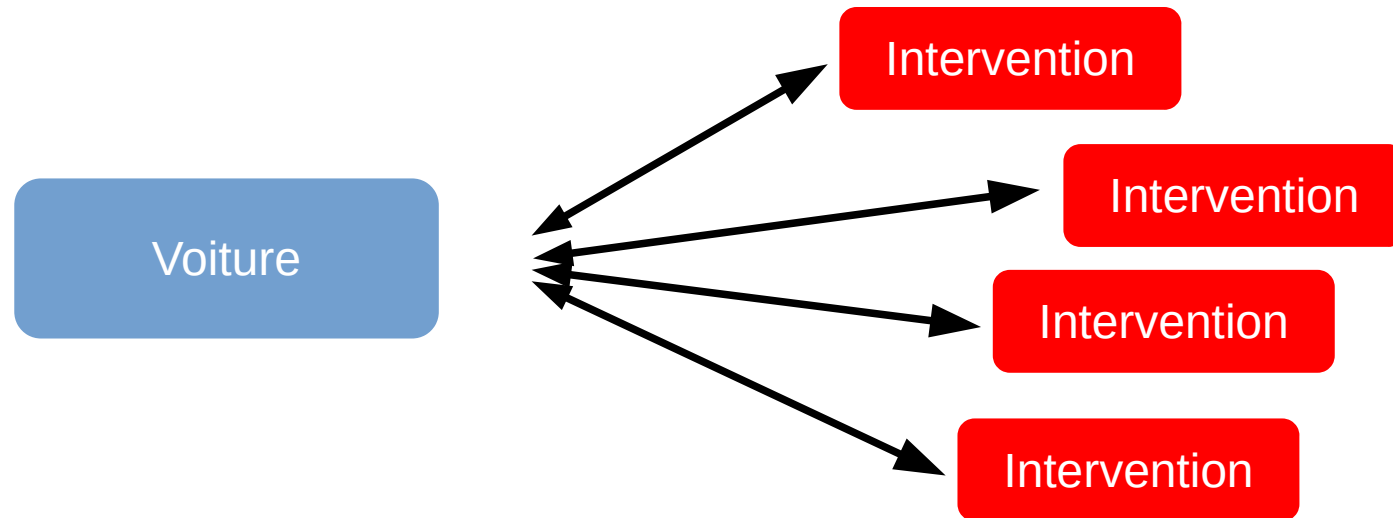
```

Les Interventions  
correspondent

# Supprimer une intervention

Attention : supprimer l'intervention ne doit pas supprimer la voiture.

Ne pas cascader la suppression




# Créons une nouvelle classe exécutable

```
public class DeleteInterventionDemo{

    public static void main(String[] args) {

        . . .
        try {
            // récupérer une session & ouvrir une transaction
            session = factory.getCurrentSession();
            session.beginTransaction();

            //commit transaction
            session.getTransaction().commit();
            System.out.println("Terminé !");
        }
        finally {
            //régler le pb de connection leak
            session.close();
            factory.close();
        }
    }
}
```



id	dateheure	prix	titre	technicien	voiture_id
1	2020-10-10 08:00:00	80.50	Petite Révision	A. Didonk	1
2	2019-12-05 13:30:00	20.00	Changement des pneus avant	M. Imome	1
3	2017-06-21 06:45:00	80.50	Vidange	S. Ouraille	1
NULL	NULL	NULL	NULL	NULL	NULL

```
public class DeleteInterventionDemo{  
    public static void main(String[] args) {  
        . . .  
        // récupérer une session & ouvrir une transaction  
        session = factory.getCurrentSession();  
        session.beginTransaction();  
  
        // récupérer une intervention  
        int id=1;  
        Intervention i = session.get(Intervention.class, id);  
  
        // supprimer une intervention  
        if(i!=null) {  
            System.out.println("Supression de l'intervention : "+i);  
            session.delete(i);  
        }  
  
        //commit transaction  
        session.getTransaction().commit();  
        System.out.println("Terminé !");  
    }  
}
```

# Resultat

```
INFO: HHH10001003: Autocommit mode: false
août 03, 2020 5:13:05 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionF
INFO: HHH000115: Hibernate connection pool size: 1 (min=1)
août 03, 2020 5:13:05 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
Hibernate: select interventi0_.id as id1_0_0_, interventi0_.dateheure as dateheur2_0_0_, interve
Suppression de l'intervention : Intervention [id=1, dateheure=2020-10-10T10:00, titre=Petite Révi
Hibernate: delete from intervention where id=?
Terminé !
août 03, 2020 5:13:06 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionF
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/bdd_05_one_to_many_t
```

	id	dateheure	prix	titre	technicien	voiture_id	
►	2	2019-12-05 13:30:00	20.00	Changement des pneus avant	M. Imome	1	
	3	2017-06-21 06:45:00	80.50	Vidange	S. Ouraille	1	
	NULL	NULL	NULL	NULL	NULL	NULL	



# On peut visualiser le résultat coté java en re-executant le code

## GetVoitureInterventionDemo.java

```
// récupérer une session & ouvrir une transaction
...

//récupérer une voiture
Long id= 1L;
Voiture v = session.get(Voiture.class , id);

// afficher la voiture
System.out.println("voiture : "+v);

//afficher les Interventions associées
System.out.println("interventions : "+v.getInterventions());

//commit transaction
...
```