École Doctorale en Sciences Economique, Gestion et Informatique
Faculté des Sciences Économiques et de Gestion de Sfax
Département Informatique

Auditoire

Mastère Professionnel Audit et Sécurité Informatique

Architectures des Systèmes d'Information (JavaEE)

Enseignante

AÏDA KHEMAKHEM

Année Universitaire

2019 – 2020

Chapitre 1 :

JPA – EJB : Héritage

I. Héritage : Stratégies de mapping O/R

II. Premier cas : une seule table !

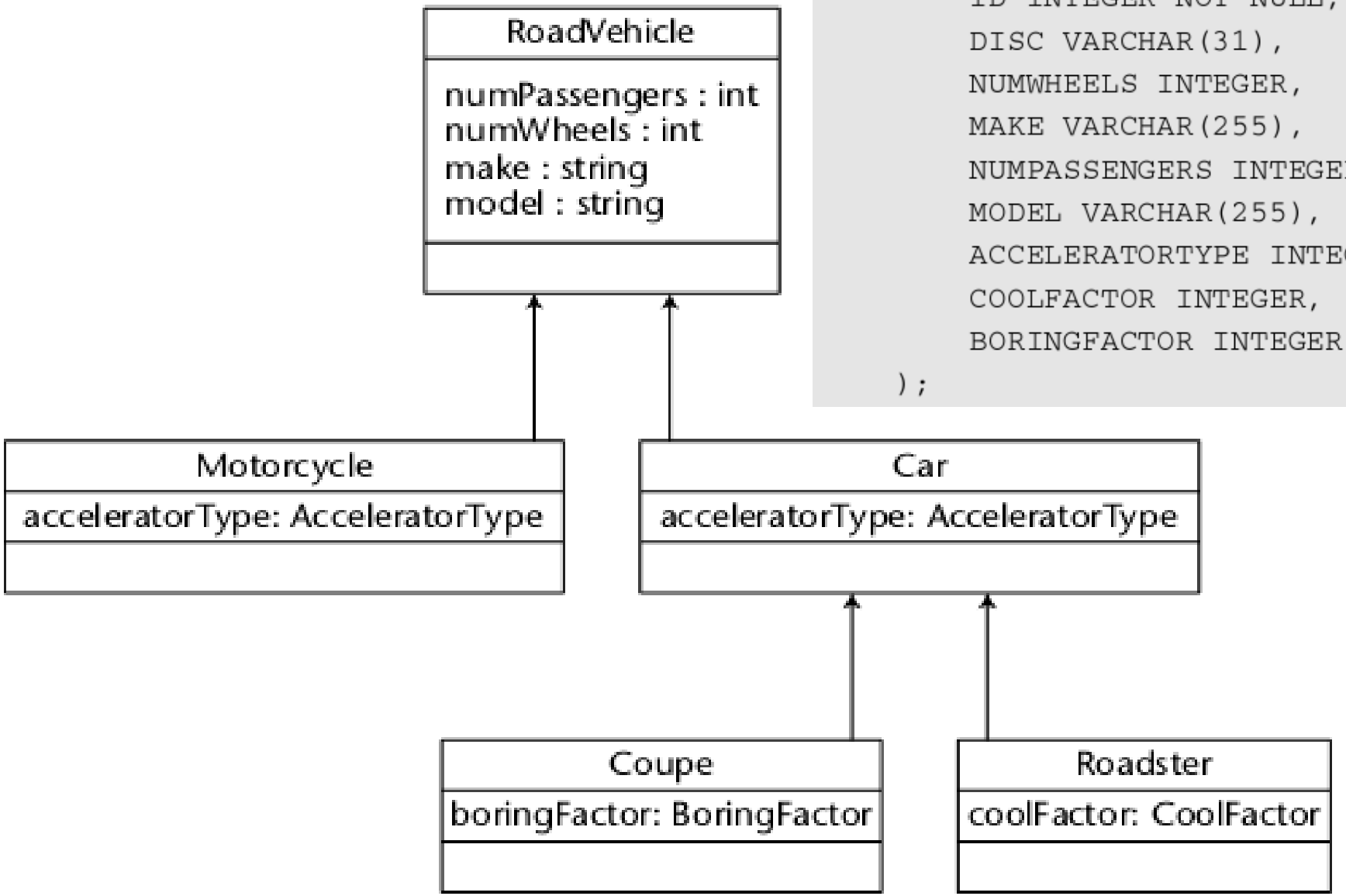
III. Deuxième stratégie : une table par classe

I. Héritage : Stratégies de mapping O/R

- Stratégies de mapping entre classes et tables quand on a de l'héritage ?
  - Une table pour toute la hiérarchie de classes ?
  - Une table par classe/sous-classe ?
  - Autres solutions ?

Exemple d'héritage : Véhicule

```
CREATE TABLE ROADVEHICLE (
        ID INTEGER NOT NULL,
        DISC VARCHAR(31),
        NUMWHEELS INTEGER,
        MAKE VARCHAR(255),
        NUMPASSENGERS INTEGER,
        MODEL VARCHAR(255),
        ACCELERATORTYPE INTEGER,
        COOLFACTOR INTEGER,
        BORINGFACTOR INTEGER
    );
```



Code de RoadVehicle.java (classe racine)

```
package examples.entity.single_table;

public class RoadVehicle {
    public enum AcceleratorType {PEDAL,THROTTLE};

    protected int numPassengers;
    protected int numWheels;
    protected String make;
    protected String model;

    // setters and getters go here
    ...

    public String toString() {
        return "Make: "+make+
            ", Model: "+model+
            ", Number of passengers: "+numPassengers;
    }
}
```

Code de Motorcycle.java

```
package examples.entity.single_table;

public class Motorcycle extends RoadVehicle {
    public final AcceleratorType acceleratorType =
        AcceleratorType.THROTTLE;

    public Motorcycle() {
        numWheels = 2;
        numPassengers = 2;
    }

    public String toString() {
        return "Motorcycle: "+super.toString();
    }
}
```

Code de Car.java

```
package examples.entity.single_table;

public class Car extends RoadVehicle {
    public final AcceleratorType acceleratorType =
        AcceleratorType.PEDAL;

    public Car() {
        numWheels = 4;
    }

    public String toString() {
        return "Car: "+super.toString();
    }
}
```

Code de Roadster.java

```
package examples.entity.single_table;

public class Roadster extends Car {
    public enum CoolFactor {COOL,COOLER,COOLEST};

    private CoolFactor coolFactor;

    public Roadster() {
        numPassengers = 2;

     }

     // setters and getters go here
     ...

     public String toString() {
         return "Roadster: "+super.toString();
     }
}
```

Code de Coupe.java

```
package examples.entity.single_table;

public class Coupe extends Car {
    public enum BoringFactor {BORING,BORINGER,BORINGEST};

    private BoringFactor boringFactor;

    public Coupe() {
        numPassengers = 5;
    }

    // setters and getters go here
    ...

    public String toString() {
        return "Coupe: "+super.toString();
    }
}
```

II. Premier cas : une seule table !

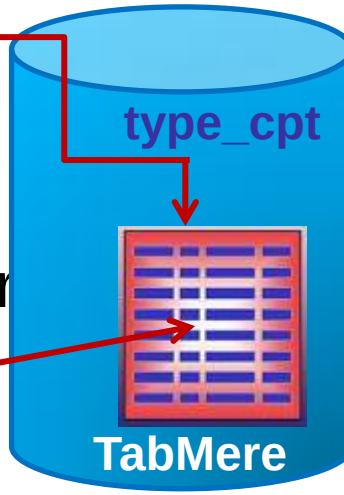
- Une seule table représente toute la hiérarchie
- Une colonne de « discrimination » est utilisée pour distinguer les sous-classes
- Cette solution supporte le polymorphisme
- Désavantages :
  - Une colonne pour chaque champ de chaque classe,
  - Comme une ligne peut être une instance de chaque classe, des champs risquent de ne servir à rien (nullable)

II.1. Les annotations d'héritage

- Au niveau de **la classe mère**
  - `@Inheritance(strategy=InheritanceType.SINGLE_TABLE)`

    Cette annotation pour spécifier stratégie de représentation de l'héritage dans la BD
  - Dans la stratégie d'une **seule table** dans a BD, il faut préciser nom de la colonne qui précise la type de l'objet, en utilisant :

```
@DiscriminatorColumn(name="type_cpt",
discriminatorType=DiscriminatorType.STRING)
```



- Au niveau de **la classe fille**, il faut préciser la valeur qui sera affectée dans la colonne type :

```
@DiscriminatorValue("classeFille")
```

La valeur de cette annotation doit être de même type précisé dans la classe mère

II.2. Exemple d'annotation pour de cas d'héritage

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="type_cpt",
discriminatorType=DiscriminatorType.STRING)
public class ClasseMere implements Serializable{
@Id
private String numero;
...
}
```

```
@Entity
@DiscriminatorValue("courant")
public class ClFille extends ClasseMere implements Serializable
{
private double decouvert;
...
}
```

Code de la classe mère avec les annotations

```
package examples.entity.single_table;

// imports go here

@Entity(name="RoadVehicleSingle")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="DISC",
    discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue("ROADVEHICLE")
public class RoadVehicle implements Serializable {
    public enum AcceleratorType {PEDAL,THROTTLE};
```

Suite de la classe mère RoadVehicule

```
    @Id
    protected int id;
    protected int numPassengers;
    protected int numWheels;
    protected String make;
    protected String model;

    public RoadVehicle() {
        id = (int) System.nanoTime();
    }

    // setters and getters go here
    ...
}
```

Code de la classe dérivée Motorcycle.java

```
package examples.entity.single_table;

// imports go here

@Entity
@DiscriminatorValue("MOTORCYCLE")
public class Motorcycle extends RoadVehicle implements Serializable {
    public final AcceleratorType acceleratorType =
            AcceleratorType.THROTTLE;

    public Motorcycle() {
        super();
        numWheels = 2;
        numPassengers = 2;
    }
}
```

Code de la classe dérivée Car.java

```
package examples.entity.single_table;

// imports go here

@Entity
@DiscriminatorValue("CAR")
public class Car extends RoadVehicle implements Serializable {
    public final AcceleratorType acceleratorType =
        AcceleratorType.PEDAL;

    public Car() {
        super();
        numWheels = 4;
    }
}
```

Code de la classe dérivée Roadster.java

```
package examples.entity.single_table;

// imports go here

@Entity
@DiscriminatorValue("ROADSTER")
public class Roadster extends Car implements Serializable {
    public enum CoolFactor {COOL,COOLER,COOLEST};

    private CoolFactor coolFactor;

    public Roadster() {
        super();
        numPassengers = 2;
    }

    // setters and getters go here
    ...
}
```

Code de la classe dérivée Coupe.java

```
package examples.entity.single_table;

// imports go here

@Entity
@DiscriminatorValue("COUPE")
public class Coupe extends Car implements Serializable {
    public enum BoringFactor {BORING,BORINGER,BORINGEST};

    private BoringFactor boringFactor;

    public Coupe() {
        super();
        numPassengers = 5;
    }

    // setters and getters go here
    ...
}
```

Table correspondante

```
CREATE TABLE ROADVEHICLE (
        ID INTEGER NOT NULL,
        DISC VARCHAR(31),
        NUMWHEELS INTEGER,
        MAKE VARCHAR(255),
        NUMPASSENGERS INTEGER,
        MODEL VARCHAR(255),
        ACCELERATORTYPE INTEGER,
        COOLFACTOR INTEGER,
        BORINGFACTOR INTEGER
    );
```

III. Deuxième stratégie : une table par classe

- Il suffit de modifier quelques annotations !
  - Dans RoadVehicle.java

```
@Entity(name="RoadVehicleJoined")
@Table(name="ROADVEHICLEJOINED")
@Inheritance(strategy=InheritanceType.JOINED)
public class RoadVehicle {
    ...
```

- On peut retirer les @Discriminator des sous-classes (on aura des valeurs par défaut)
- Le champ Id de la classe RoadVehicle sera une clé étrangère dans les tables des sous-classes,
- Remarque : on utilise ici @TABLE pour ne pas que la table porte le même nom que dans l'exemple précédent (facultatif)

Les tables !

Table 9.2 ROADVEHICLEJOINED Table

ID	DTYPE	NUMWHEELS	MAKE	MODEL
1423656697	Coupe	4	Bob	E400
1425368051	Motorcycle	2	NULL	NULL
1424968207	Roadster	4	Mini	Cooper S

Table 9.3 MOTORCYCLE Table

ID	ACCELERATORTYPE
1425368051	1

Les tables (suite)

Table 9.4 CAR Table

ID	ACCELERATORTYPE
1423656697	0
1424968207	0

Table 9.5 COUPE Table

ID	BORINGFACTOR
1423656697	0

Table 9.6 ROADSTER Table

ID	COOLFACTOR
1423656697	2