

Lien SGBD, Langage OO

JPA et Hibernate

Donatello Conte

4 novembre 2019

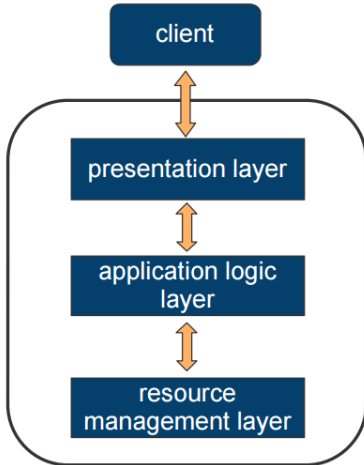
Plan

- 1 Motivation
- 2 JPA
- 3 Bases du Mapping OOR
- 4 Mapping des relations
- 5 Mapping de l'héritage
- 6 Gestion de la persistance
- 7 Récupération de données

Plan

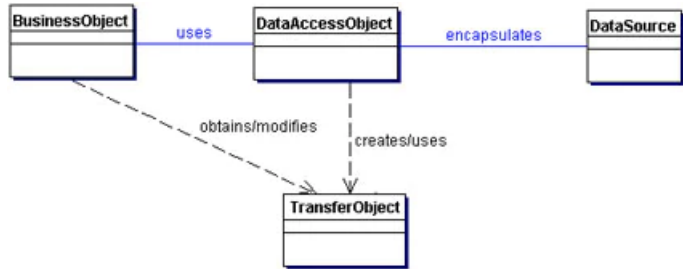
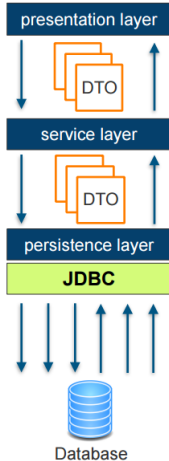
- 1 Motivation
- 2 JPA
- 3 Bases du Mapping OOR
- 4 Mapping des relations
- 5 Mapping de l'héritage
- 6 Gestion de la persistance
- 7 Récupération de données

Architecture en couches d'un système d'information



- Couche Présentation
 - interface de communication à des entités externes
 - vue dans le modèle MVC
- Couche Métier
 - opérations demandées par le client à travers la couche de présentation
- Couche de gestion des ressources (couche de persistance)
 - sources de données
 - responsable du stockage et de la récupération des données

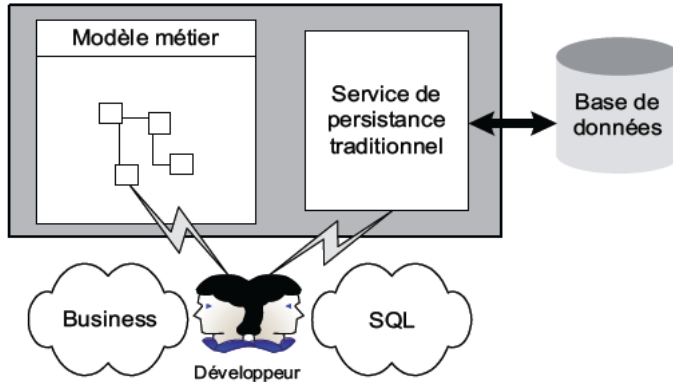
Modèle traditionnel avec JDBC



JDBC : inconvénients

- Il faut beaucoup de code "inutile" pour les différentes opérations CRUD (Create Request Update Delete)
- Mapping manuel entre les résultats des requêtes et les classes Java
- Pas de support d'association entre les classes, de l'héritage, du polymorphisme

La persistance non transparente



La persistance non transparente

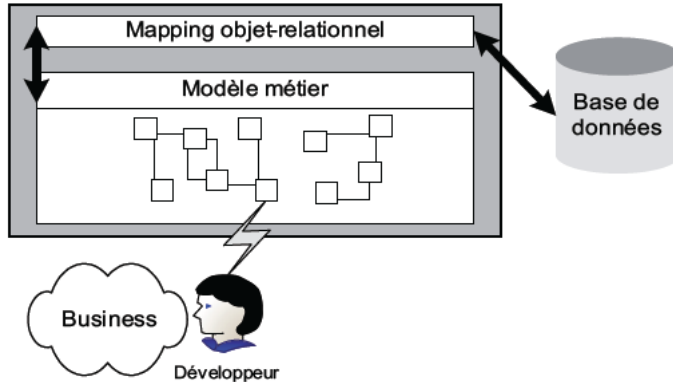
```
public Langage create(Langage obj) {
    try {

        //Vu que nous sommes sous postgres, nous allons chercher manuellement
        //la prochaine valeur de la séquence correspondant à l'id de notre table
        ResultSet result = this .connect
            .createStatement(
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_UPDATABLE
            ).executeQuery(
                "SELECT NEXTVAL('langage_lan_id_seq') as id"
            );
        if(result.first()){
            long id = result.getLong("id");
            PreparedStatement prepare = this .connect
                .prepareStatement(
                    "INSERT INTO langage (lan_id, lan_nom) VALUES(?, ?)"
                );
            prepare.setLong(1, id);
            prepare.setString(2, obj.getNom());

            prepare.executeUpdate();
            obj = this.find(id);

        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return obj;
}
```


La persistance transparente



La persistance transparente

```
public void persist(Book entity) {  
    getSession().save(entity);  
}
```

Le mapping objet-relationnel

- Consiste à décrire une correspondance entre un schéma de base de données et un modèle de classes
- Permet de se focaliser sur les aspects métier de l'application
- Permet une synchronisation automatique entre les objets et la base de données
- Portabilité augmentée
- Permet de faire des requêtes à un niveau d'abstraction supérieur

Plan

- 1 Motivation
- 2 JPA
- 3 Bases du Mapping OOR
- 4 Mapping des relations
- 5 Mapping de l'héritage
- 6 Gestion de la persistance
- 7 Récupération de données

JPA

- Spécification pour la gestion de la persistance et du mapping objet / relationnel avec Java
- Persistance : Les objets survivent à l'arrêt de la JVM
- Objectif : fournir un outil de mapping Objet / Relationnel pour les développeurs Java utilisant un modèle de domaine Java et une base de données relationnelle
- Hibernate : implémentation complète des spécifications JPA

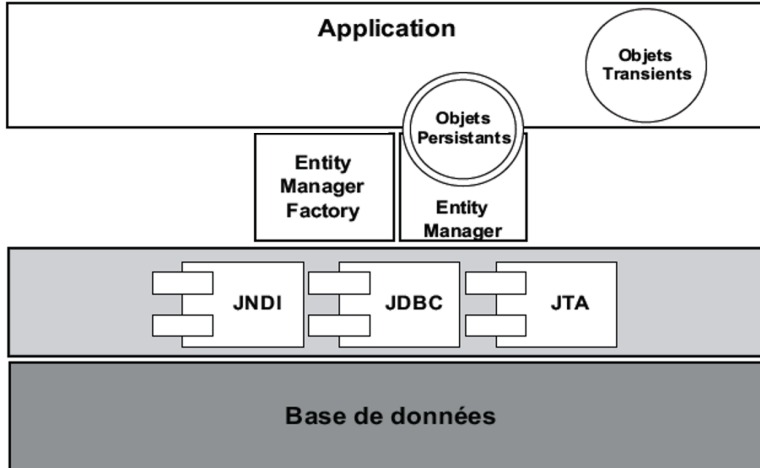
ORM et JPA

- Avec JPA/Hibernate il semble y avoir quelque chose de "magique" entre les objets et la base de données
- Au coeur de tout ça il y a la mapping objects/relationnel
- On peut vérifier quel sont les opérations SQL réellement exécutées

Plan

- 1 Motivation
- 2 JPA
- 3 Bases du Mapping OOR**
- 4 Mapping des relations
- 5 Mapping de l'héritage
- 6 Gestion de la persistance
- 7 Récupération de données

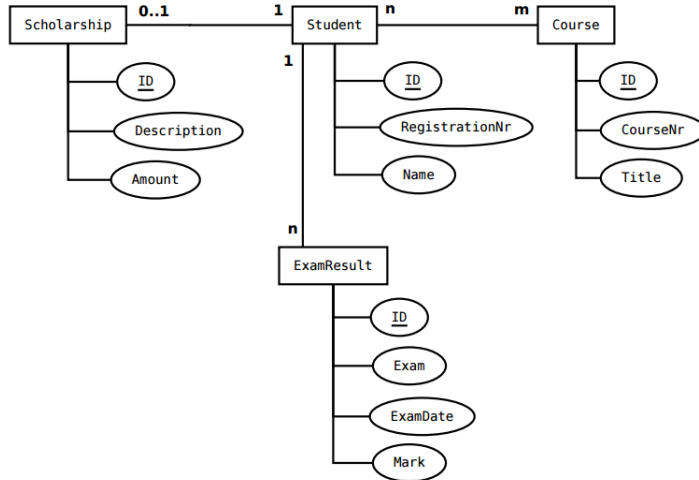
Architecture JPA



Entités

- POJO
- Elles représentent généralement une table dans une base de données relationnelle
- Chaque instance de l'entité correspond à une ligne dans cette table
- Elles doivent avoir une identité persistante
- Elles peuvent avoir à la fois des données persistantes et transitoires (Transient en anglais)

Un modèle de classes



Exemple de mapping simple

```
@Entity
public class ExamResult {
    @Id
    private Long id;

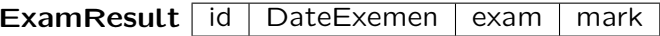
    @Column(name = "DateExemen")
    @Temporal(TemporalType.DATE)
    private Date examDate;

    private String exam;

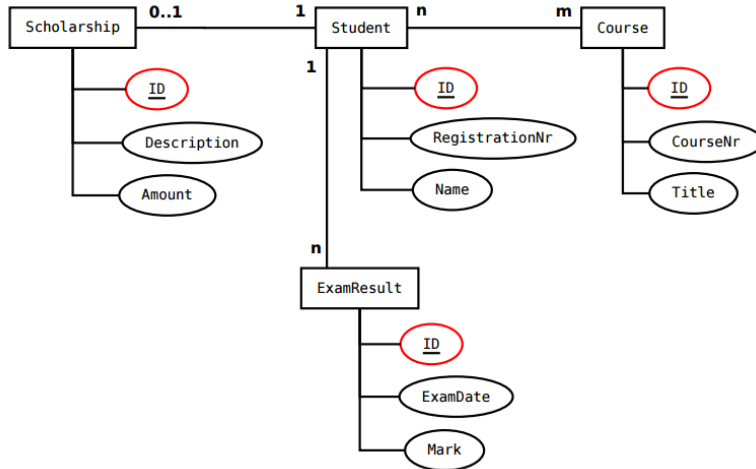
    private int mark;

    @Transient
    private String examLocation;
    ...
}
```

Annotation	Description
@Entity	La classe est une entité
@Id	Spécifies la clé primaire de l'entité
@Temporal	Utilisé pour les variables de type java.util.Date et java.util.Calendar
@Transient	Spécifies que la variable n'est pas persistante



Heritage



Heritage

```
@MappedSuperclass
public class BaseEntity {
    @Id
    @GeneratedValue(strategy =
        GenerationType.AUTO)
    protected Long id;
    public Long getId() { return id;
    }
}

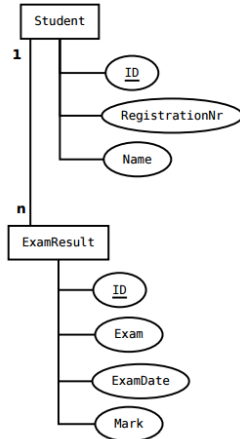
@Entity
public class ExamResult extends
    BaseEntity {
    @Column(name = "DateExemen")
    @Temporal(TemporalType.DATE)
    private Date examDate;
    private String exam;
    private int mark;
    @Transient
    private String examLocation;
    //Getter and setters omitted
}
```

Annotation	Description
@MappedSuperclass	Il n'y a pas une table correspondante mais le mapping de ses variables sera appliqué aux sous-classes
@GeneratedValue	Spécifies comment générer une nouvelle valeur (typiquement pour les clé primaires)

ExamResult

id	DateExemen	exam	mark
----	------------	------	------

Orienté-objets vs SQL



```
public class Student extends BaseEntity {
    private String registrationNumber;
    private String name;
    private List<ExamResult> examResults;
    ...
}
```

```
public class ExamResult extends BaseEntity {
    private Date examDate;
    private String exam;
    private int mark;
    private Student student;
    ...
}
```

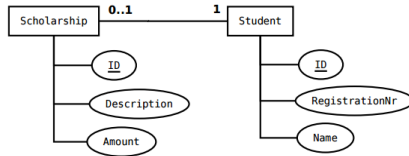
Plan

- 1 Motivation
- 2 JPA
- 3 Bases du Mapping OOR
- 4 Mapping des relations**
- 5 Mapping de l'héritage
- 6 Gestion de la persistance
- 7 Récupération de données

Relations entre entités

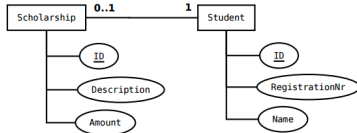
- One-to-one, one-to-many, many-to-many, many-to-one
- Unidirectionnelles, bidirectionnelles
- Représentées à travers des Collection (List, Set, Map, etc.)
- Nécessite de spécifier "le propriétaire" de la relation

Mapping one-to-one



- 1 Utilisation d'une seule table où Scholarship est la table intégrée
- 2 Deux tables séparées. La clé primaire de Scholarship est une clé étrangère vers Student (qui est le propriétaire de la relation)
- 3 Deux tables séparées. Student contient la clé étrangère vers Scholarship. La clé étrangère a une contrainte **unique**.
- 4 Deux tables séparées. Scholarship contient la clé étrangère vers Student. La clé étrangère a une contrainte **unique**.

Mapping One-to-One



```

CREATE TABLE EMBEDDEDSTUDENT
(
  ID BIGINT PRIMARY KEY NOT
    NULL,
  NAME VARCHAR(255),
  NUMEROETUDIANT VARCHAR(255),
  AMOUNT INTEGER,
  DESCRIPTION VARCHAR(255)
);
CREATE UNIQUE INDEX
  anIndexName
ON EMBEDDEDSTUDENT (
  NUMEROETUDIANT );
  
```

@Entity

```

public class EmbeddedStudent extends BaseEntity {
  @Column(name = "numeroEtudiant", unique = true)
  private String registrationNumber;
  private String name;
  @Embedded
  private EmbeddedScholarship scholarship;
  @Transient
  private DateTime loginTime;
  ...
}
  
```

@Embeddable

```

public class EmbeddedScholarship {
  private String description;
  private Integer amount;
}
  
```

Mapping One-to-One

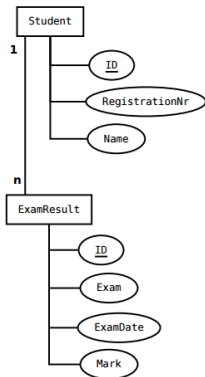
```
CREATE TABLE SCHOLARSHIP
(
    ID BIGINT PRIMARY KEY NOT NULL,
    AMOUNT INTEGER,
    DESCRIPTION VARCHAR(255),
    STUDENT_ID BIGINT,
    FOREIGN KEY ( STUDENT_ID )
        REFERENCES STUDENT ( ID )
);
CREATE UNIQUE INDEX uniqueIndexName
ON SCHOLARSHIP ( STUDENT_ID );

CREATE TABLE STUDENT
(
    ID BIGINT PRIMARY KEY NOT NULL,
    NAME VARCHAR(255),
    NUMEROETUDIANT VARCHAR(255)
);
CREATE UNIQUE INDEX anIndexName ON
    STUDENT ( NUMEROETUDIANT );
```

```
@Entity
public class Student extends BaseEntity {
    @Column(name = "numeroEtudiant", unique = true)
    private String registrationNumber;
    private String name;
    @OneToOne(fetch = FetchType.LAZY,
        cascade = CascadeType.ALL,
        mappedBy="grantedTo")
    private Scholarship scholarship;
    @Transient
    private DateTime loginTime;
}

@Entity
public class Scholarship extends BaseEntity {
    private String description;
    private Integer amount;
    @JoinColumn(name="student_id", unique=true)
    @OneToOne
    private Student grantedTo;
}
```

Mapping One-to-Many



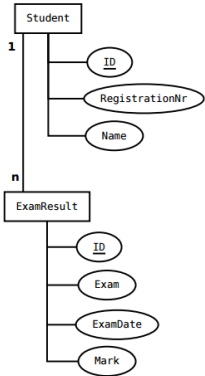
```

@Entity
public class Student extends BaseEntity {
    @Column(name = "numeroEtudiant", unique = true)
    private String registrationNumber;
    private String name;
    @OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL,
        mappedBy = "grantedTo")
    private Scholarship scholarship;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "student")
    private List<ExamResult> examResults;
    @Transient
    private DateTime loginTime;
}
  
```

```

CREATE TABLE STUDENT
(
    ID BIGINT PRIMARY KEY NOT NULL,
    NAME VARCHAR(255),
    NUMEROETUDIANT VARCHAR(255)
);
CREATE UNIQUE INDEX anIndexNameB ON STUDENT ( NUMEROETUDIANT );
  
```

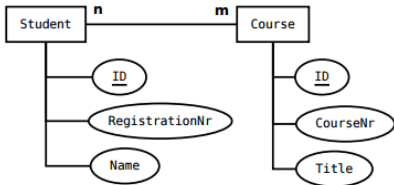
Mapping One-to-Many



```
@Entity
public class ExamResult extends BaseEntity {
    @Column(name = "dateExamen") @Temporal(TemporalType.DATE)
    private Date examDate;
    private String exam;
    private int mark;
    @ManyToOne
    private Student student;
    @Transient
    private String examLocation;
}
```

```
CREATE TABLE EXAMRESULT
(
  ID BIGINT PRIMARY KEY NOT NULL,
  EXAM VARCHAR(255),
  DATEEXAMEN DATE,
  MARK INTEGER NOT NULL,
  STUDENT_ID BIGINT,
  FOREIGN KEY ( STUDENT_ID ) REFERENCES STUDENT ( ID )
);
```

Mapping Many-to-Many

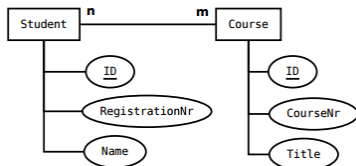


```
@Entity
public class Student extends BaseEntity {
    private String name;
    ...
    @OneToOne(fetch = FetchType.LAZY, cascade =
        CascadeType.ALL, mappedBy = "grantedTo")
    private Scholarship scholarship;

    @ManyToMany(mappedBy = "students")
    private List<Course> courses;
}
```

```
CREATE TABLE STUDENT
(
    ID BIGINT PRIMARY KEY NOT NULL,
    NAME VARCHAR(255),
    NUMEROETUDIANT VARCHAR(255)
);
CREATE UNIQUE INDEX anIndexNameB ON STUDENT (
    NUMEROETUDIANT );
```

Mapping Many-to-Many



```

@Entity
public class Course extends BaseEntity {
    private String courseNumber;
    private String title;
    @ManyToMany
    @JoinTable(name="COURSE_STUDENT",
        joinColumns={@JoinColumn(name="COURSES_ID",
            referencedColumnName="ID")},
        inverseJoinColumns={@JoinColumn(name="STUDENTS_ID",
            referencedColumnName="ID")})
    private List<Student> students;
}
  
```

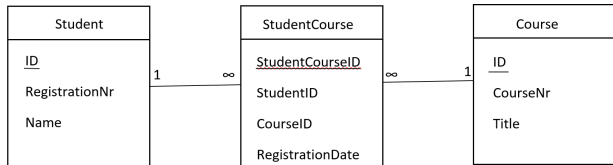
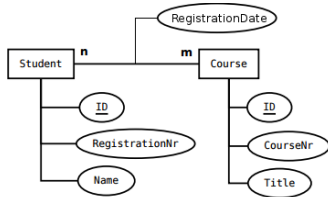
```

CREATE TABLE COURSE (
    ID BIGINT PRIMARY KEY
    NOT NULL,
    COURSENUMBER
    VARCHAR(255),
    TITLE VARCHAR(255) );
  
```

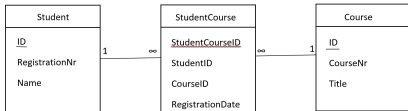
```

CREATE TABLE COURSE_STUDENT (
    COURSES_ID BIGINT NOT NULL,
    STUDENTS_ID BIGINT NOT NULL,
    FOREIGN KEY ( COURSES_ID ) REFERENCES COURSE ( ID ),
    FOREIGN KEY ( STUDENTS_ID ) REFERENCES STUDENT ( ID ) );
  
```

Mapping Many-to-Many avec attributs



Mapping Many-to-Many avec attributs



```

@Entity
public class Student {
    ...
    @OneToMany(mappedBy =
        "student")
    private Set<StudentCourse>
        studentCourse = new
        HashSet<StudentCourse>();
}

```

```

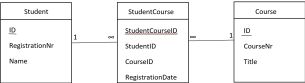
@Entity
public class Course {
    ...
    @OneToMany(mappedBy = "course")
    private Set<StudentCourse> StudentCourse = new
        HashSet<StudentCourse>();
}

@Entity
public class StudentCourse {
    @Id
    private long student_course_id;
    @ManyToOne(cascade = CascadeType.ALL)
    private Student student;
    @ManyToOne(cascade = CascadeType.ALL)
    private Course course;

    // additional fields
    private Date registeredDate;
}

```

Mapping Many-to-Many avec attributs



```
CREATE TABLE student (
    id int(11) NOT NULL AUTO_INCREMENT,
    registrationnr varchar(15) NOT NULL,
    name varchar(45) NOT NULL,
    PRIMARY KEY (id));
CREATE TABLE course (
    id int(11) NOT NULL AUTO_INCREMENT,
    coursennr varchar(15) NOT NULL,
    title varchar(45) NOT NULL,
    PRIMARY KEY (id));
CREATE TABLE student_course (
    student_course_id int(11) NOT NULL AUTO_INCREMENT,
    student_id int(11) NOT NULL,
    course_id int(11) NOT NULL,
    registrationdate datetime NOT NULL,
    PRIMARY KEY (user_group_id),
    KEY fk_student (student_id),CONSTRAINT fk_student FOREIGN
    KEY (student_id) REFERENCES course(id),
    KEY fk_course (group_id),CONSTRAINT fk_course FOREIGN KEY
    (course_id) REFERENCES student(id));
```

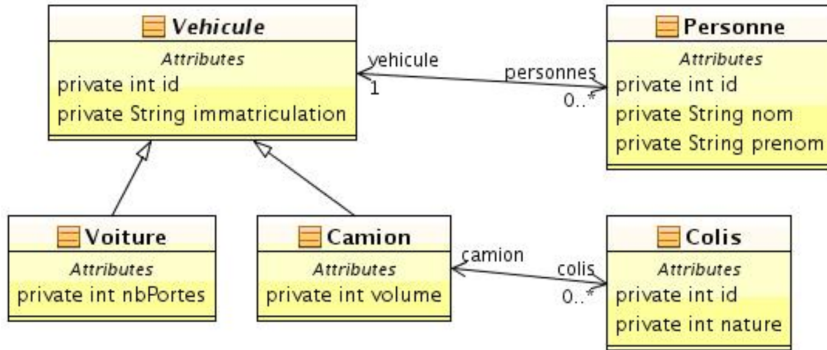
Cascade et Fetch

- Types de cascade
 - **ALL, PERSIST, MERGE, REMOVE, REFRESH, DETACH**
 - Pour spécifier les opérations en cascade pour les entités associées
- Stratégies de chargement (Fetch)
 - Défini comment les hiérarchies d'objets sont chargés
 - **EAGER** : charge tous les objets liés immédiatement
 - **LAZY** : charger les objets liés à la demande (quand il y a le premier accès)

Plan

- 1 Motivation
- 2 JPA
- 3 Bases du Mapping OOR
- 4 Mapping des relations
- 5 Mapping de l'héritage**
- 6 Gestion de la persistance
- 7 Récupération de données

Exemple d'héritage



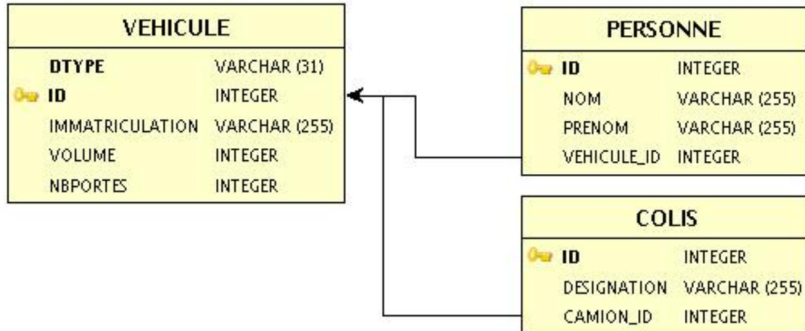
En Java

```
@Entity
public abstract class Vehicule {
    @Id @GeneratedValue Integer id;
    String immatriculation;
    ...
}

@Entity
public class Voiture extends Vehicule {
    int nbPortes;
    ...
}

@Entity
public class Camion extends Vehicule {
    int volume;
    ...
}
```

Une seule table



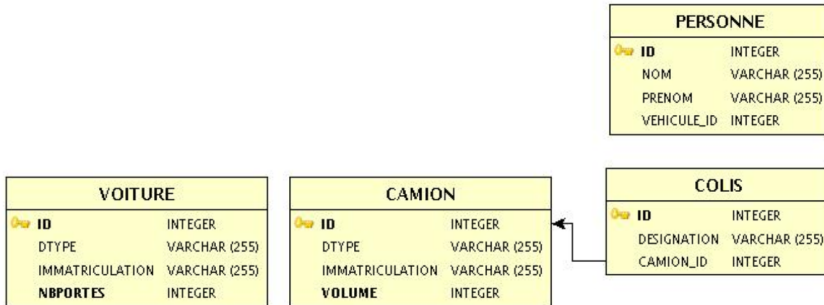
En Java

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
    @DiscriminatorColumn(name="TYPE_VEHICULE")
public abstract class Vehicule { ... }

@Entity
@DiscriminatorValue("V")
public class Voiture extends Vehicule { ... }

@Entity
@DiscriminatorValue("C")
public class Camion extends Vehicule { ... }
```

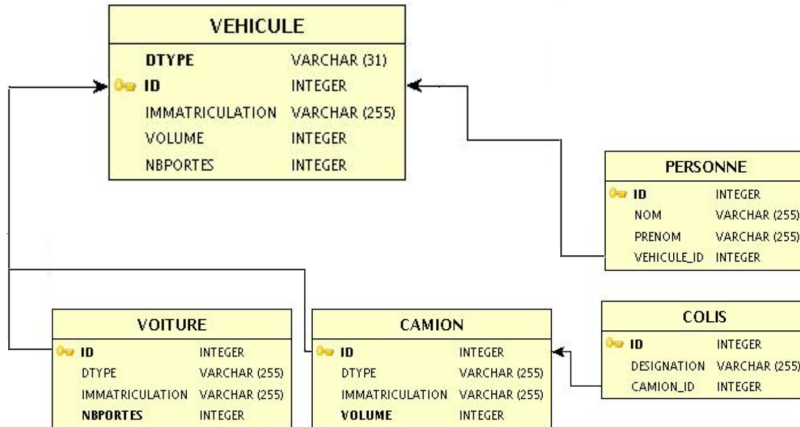
Une table par classe concrète



En Java

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public abstract class Vehicule {
    ...
}
```

Une table et une jointure par classe



En Java

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public abstract class Vehicule {
    ...
}
```

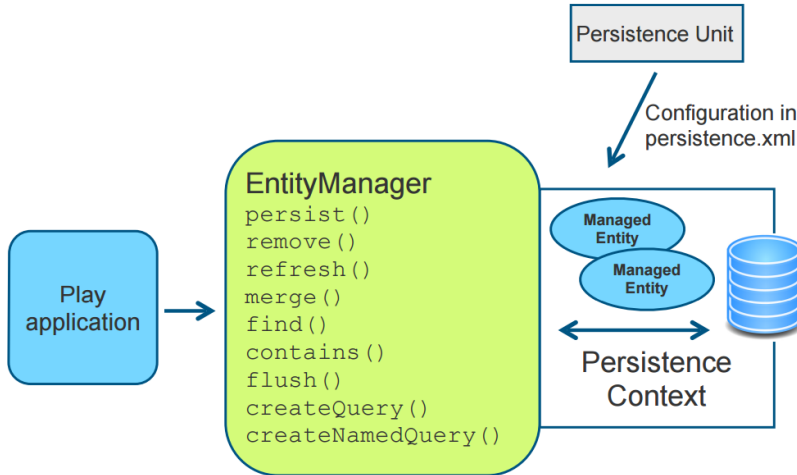
Plan

- 1 Motivation
- 2 JPA
- 3 Bases du Mapping OOR
- 4 Mapping des relations
- 5 Mapping de l'héritage
- 6 Gestion de la persistance**
- 7 Récupération de données

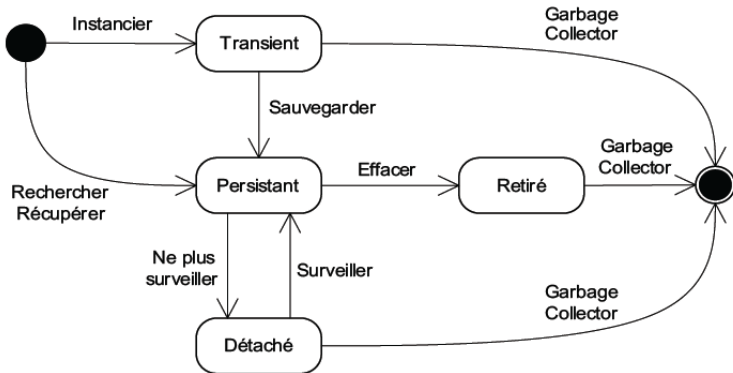
Concepts liés à la persistance

- Unité de Persistance (Persistence Unit - PU)
 - Définit un ensemble de classes d'entités gérées par l'instance Entity Manager dans une application
 - Effectue le mapping OOR
 - Définition de l'unité de persistance dans le fichier `persistence.xml`
- Contexte de persistance (Persistence Context - PC)
 - Ensemble d'instances d'entités gérées
 - Lié au contexte d'exécution
- Gérant d'entités (Entity Manager (EM))
 - API pour l'interaction avec le contexte de persistance
 - Manipule et contrôle le cycle de vie d'un contexte de persistance
 - Crée et supprime des instances d'entités persistantes
 - Exécute des requêtes sur les entités

Entity Manager



Cycle de vie d'un objet manipulé avec le gestionnaire d'entités

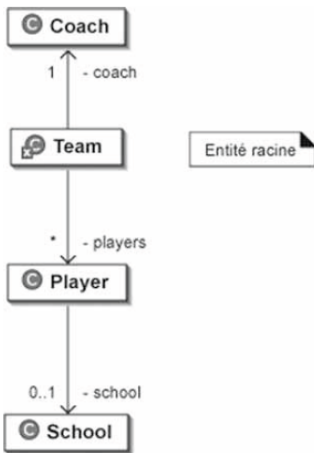


Insertion de données dans la BD

```
Employee emp = new Employee(ID, "John Doe");  
em.persist(emp);
```

- Le mot clé **new** ne suffit pas à insérer les données dans la BD
- C'est l'appel à la méthode **persist()** de l'EntityManager qui le rend persistante
- La méthode insère une nouvelle ligne dans la table correspondante
- L'EntityManager surveille l'entité pour qu'elle soit synchronisée avec la BD

Insertion d'une hiérarchie de classes



Insertion d'une hiérarchie de classes

```
@Entity
public class Team {
    @Id
    @GeneratedValue(
        strategy=GenerationType.AUTO)
    private int id;
    @OneToMany(mappedBy = "team")
    private Collection<Player> players = new ArrayList<Player>();
    @OneToOne
    private Coach coach;
    ...
}
```

Insertion d'une hiérarchie de classes

```
@Entity
public class Player {
    @Id
    @GeneratedValue(
        strategy=GenerationType.AUTO)
    private int id;
    private float height;
    private String name;
    @ManyToOne
    private Team team;
    ...
}
```

Insertion d'une hiérarchie de classes

```
...
tm.begin();
Team team = new Team("cascade test team");
Player player = new Player ("cascade player test");
School school = new School ("cascade school test");
Coach coach= new Coach ("cascade test coach");
player.setSchool(school);
team.getPlayers().add(player);
team.setCoach(coach);
em.persist(team);
em.persist(coach);
em.persist(school);
em.persist(player);
tm.commit();
...
```

Insertion d'une hiérarchie de classes

```
insert into Team (id, coach_id, name) values (null, ?, ?)
insert into Coach (id, name) values (null, ?)
insert into School (id, name) values (null, ?)
insert into Player (id, height, name, school_id) values (null, ?, ?, ?)

update Team set coach_id=?, name=? where id=?
update Player set Team_ID=? where id=?
```

Insertion d'une hiérarchie de classes

```
...
tm.begin();
Team team = new Team("cascade test team");
Player player = new Player ("cascade player test");
School school = new School ("cascade school test");
Coach coach= new Coach ("cascade test coach");
player.setSchool(school);
team.getPlayers().add(player);
team.setCoach(coach);
em.persist(team);
//em.persist(coach);
em.persist(school);
em.persist(player);
tm.commit();
...
```

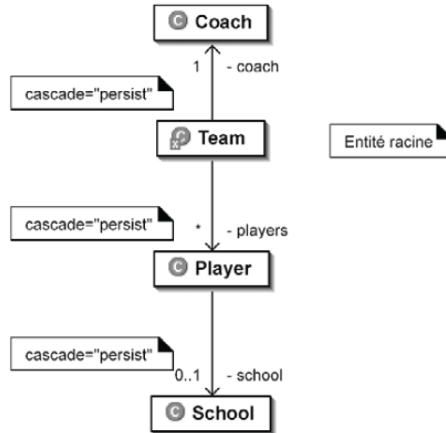
Insertion d'une hiérarchie de classes

Caused by: org.hibernate.TransientObjectException: object references an unsaved
transient instance - save the transient instance before flushing:
Team.coach -> Coach

at org.hibernate.engine.CascadingAction\$9.noCascade(CascadingAction.java:353)

at org.hibernate.engine.Cascade.cascade(Cascade.java:139)

Insertion d'une hiérarchie de classes



Insertion d'une hiérarchie de classes

```
@Entity
public class Team {
    @Id
    @GeneratedValue(
        strategy=GenerationType.AUTO)
    private int id;
    @OneToMany(mappedBy = "team", cascade=CascadeType.PERSIST)
    private Collection<Player> players = new ArrayList<Player>();
    @OneToOne(cascade=CascadeType.PERSIST)
    private Coach coach;
    ...
}
```

Insertion d'une hiérarchie de classes

```
@Entity
public class Player {
    @Id
    @GeneratedValue(
        strategy=GenerationType.AUTO)
    private int id;
    private float height;
    private String name;
    @ManyToOne(cascade=CascadeType.PERSIST)
    private Team team;
    ...
}
```

Insertion d'une hiérarchie de classes

```
...
tm.begin();
Team team = new Team("cascade test team");
Player player = new Player ("cascade player test");
School school = new School ("cascade school test");
Coach coach= new Coach ("cascade test coach");
player.setSchool(school);
team.getPlayers().add(player);
team.setCoach(coach);
em.persist(team);
tm.commit();
...
```

Insertion d'une hiérarchie de classes

```
insert into Team (id, coach_id, name) values (null, ?, ?)
insert into Coach (id, name) values (null, ?)
insert into School (id, name) values (null, ?)
insert into Player (id, height, name, school_id) values (null, ?, ?, ?)

update Team set coach_id=?, name=? where id=?
update Player set Team_ID=? where id=?
```

Modification d'instances persistantes

- Instances attachées (est présente dans le gestionnaire d'entités)
 - La moindre modification d'une propriété persistante est synchronisée en toute transparence avec la base de données
- Instances détachée (n'est pas présente dans le gestionnaire d'entités)
 - Il faut un mécanisme pour "ré-associer" l'état d'une entité détachée à un gestionnaire d'entités

```
tm.begin();
Coach coach= new Coach ("test coach");
em.persist(coach);
tm.commit();
// instance détachée
coach.setName("new name");
tm.begin();
Coach attachedCoach = em.merge(coach);
assertTrue(em.contains(attachedCoach));
assertFalse(em.contains(coach));
tm.commit();
```

Suppression de données dans la BD

```
tm.begin();  
Team t = em.find(Team.class, new Integer(1));  
em.remove(t);  
tm.commit();
```

```
update Player set Team_ID=null where Team_ID=?  
delete from Team where id=?
```

Plan

- 1 Motivation
- 2 JPA
- 3 Bases du Mapping OOR
- 4 Mapping des relations
- 5 Mapping de l'héritage
- 6 Gestion de la persistance
- 7 Récupération de données**

La fonction `find()` et le *lazy loading*

```
public <T> T find(Class<T> entityClass, Object primaryKey);
```

```
@Entity
public class Team {
    @Id
    @GeneratedValue(
        strategy=GenerationType.AUTO)
    @Column(name="TEAM_ID")
    private int id;
    @OneToMany
    private Set<Player> players = new HashSet<Player>();
    ...
}
```

```
Team team = (Team) em.find(Team.class, new Integer(1));
System.out.println(((Player) team.getPlayers().iterator().next()).getName());
```

64/71

65/71

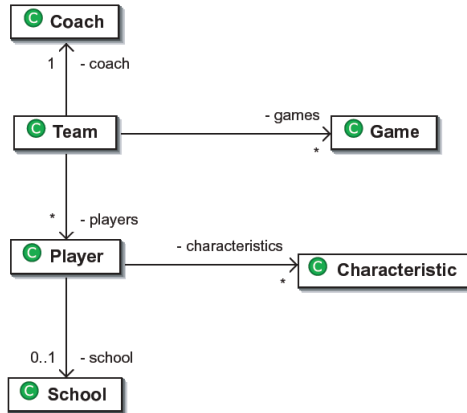
La fonction `createQuery()` et le JPA-QL

- JPQ-QL : une encapsulation du SQL selon une logique orientée objet

```
String queryString = "requête en JPA-QL";
Query query = em.createQuery(queryString);
```

```
List results = query.getResultList();
Object o = query.getSingleResult();
```

JPA-QL



JPA-QL

```
select team.name, team.id from Team team
```

```
String queryString = "select team.name, team.id from Team team ";
Query query = em.createQuery(queryString);
List results = query.getResultList();
// pour chaque ligne de résultat, nous avons deux éléments
// dans le tableau d'objets
Object[] firstResult = (Object[])results.get(0);
String firstTeamName = (String)firstResult[0];
Long firstTeamId = (Long)firstResult[1];
```

JPA-QL

```
select team, player from Team team, Player player
```

```
String queryString = "select team, player from Team team, Player player";
Query query = em.createQuery(queryString);
List results = query.getResultList();
Object[] firstResult = (Object[])results.get(0);
Team firstTeam = (Team)firstResult[0];
Player firstPlayer = (Player)firstResult[1];
```

from Team team, Coach coach

équivalent à

Select team, coach from Team team, Coach coach

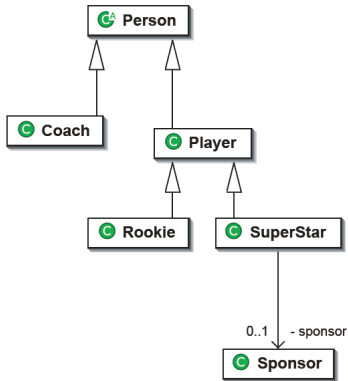
JPA-QL

```
Select team.coach from Team team
```

```
Select player.team.coach from Player p
```

```
select coach
from Player player
join player.team team
join team.coach coach
```

JPA-QL : polymorphisme nativement supporté par les requêtes



from Person p

- Cette requête retourne les instances de **Coach**, **Player**, **Rookie** et **SuperStar** (**Person** étant une classe abstraite)