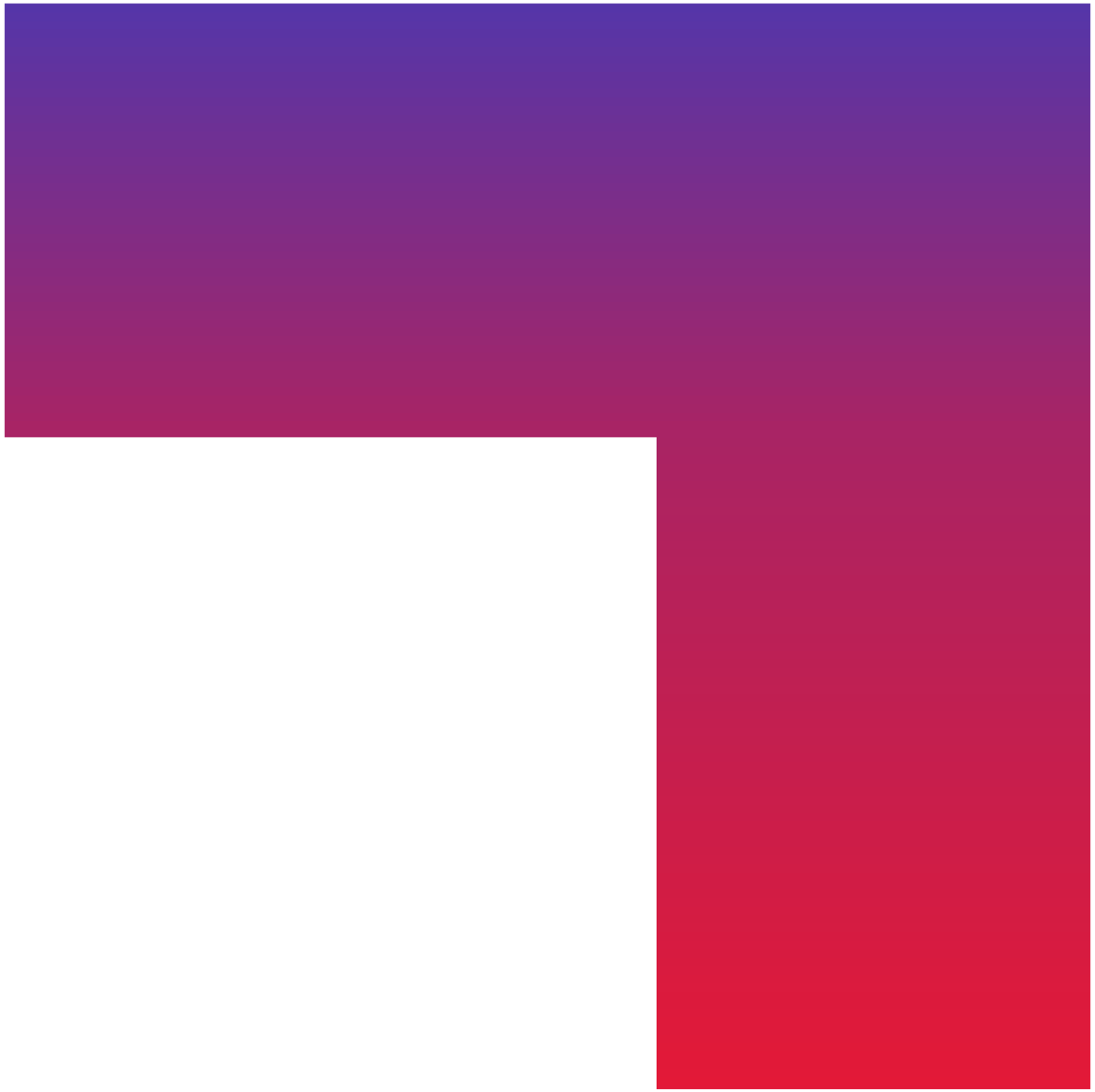
JPA

Java Persistence API



Sébastien Chassande-Barrioz
sebastien.chassande-barrioz@cgi.com

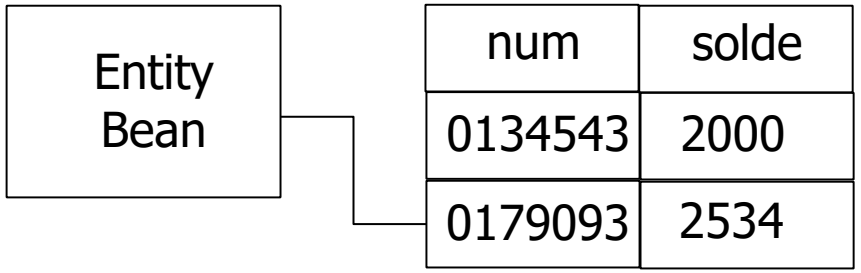
CGI

http://tinyurl.com/formation-ecom

JPA : introduction

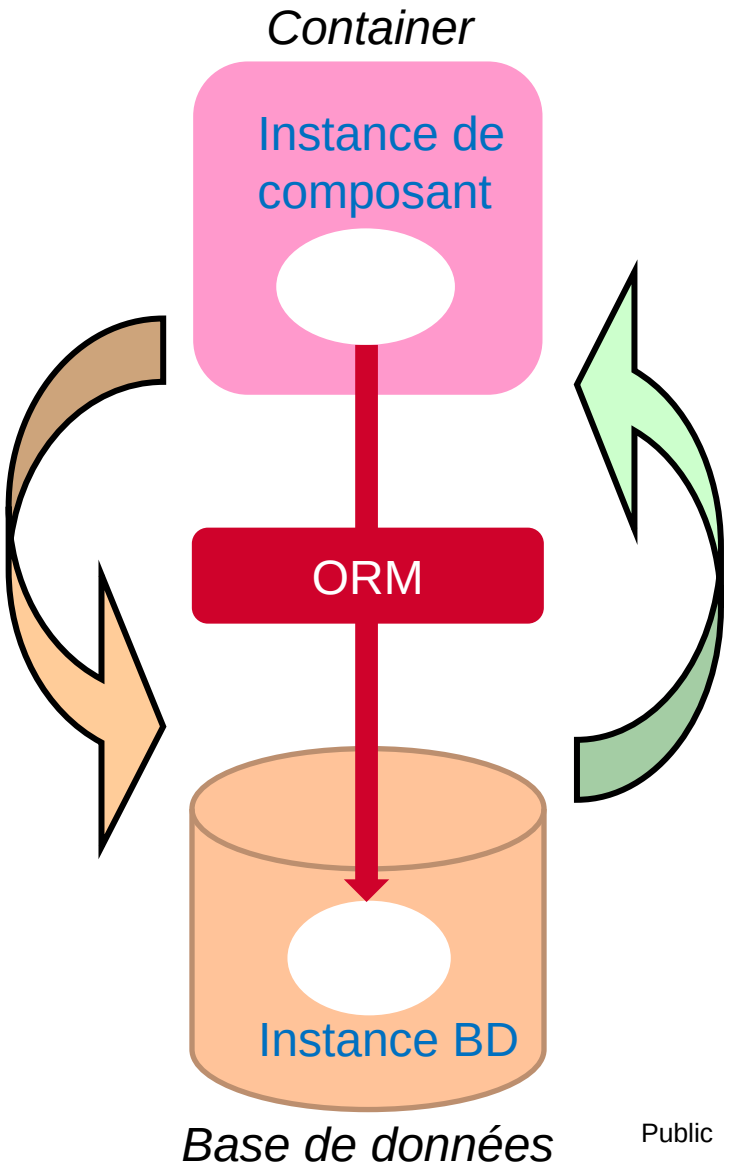
- Représentation d'une donnée manipulée par l'application
  - Donnée typiquement stockée dans un SGBD (ou tout autre support accessible en JDBC)
  - Correspondance objet – tuple relationnel (*mapping O/R*)
  - Possibilité de définir des clés, des relations, des recherches
- Avantage : manipulation d'objets Java plutôt que de requêtes SQL

- Mis en œuvre à l'aide
  - d'annotations Java 5
  - de la généricité Java 5
  - de l'API JPA (Java Persistence API)
- Implémentation: Hibernate, EclipseLink, Data Nucléus …



Modèle de persistance des entités

- Entity bean = POJO
- Notion de « clé primaire »
  - Nom persistant qui désigne 1 instance BD
- Sélection d'entités persistantes
  - Annotation *NamedQuery*
  - Implémentées par une requête JPAQL
  - Retourne une entité ou une collection d'entités
- ORM : Object Relational Mapping



Programmation d'un Entity

- POJO avec annotations (javax.persistence.*)
- Annotations de classes
  - @Entity pour définir une classe correspondant à un bean entité
- Annotations de méthode ou d'attribut
  - @Id pour définir une clé primaire (**Obligatoire !**)

```
import javax.persistence.*;
…
@Entity
public class Participant{

  @Setter @Getter
  @Id
  private long id;

  @Setter @Getter
  private String name
}
```

Identifiant Auto généré par une séquence en BD

L'identifiant peut être généré par :

- L'application
- Valeur max de la colonne + 1
- Une séquence
- L'ORM

```
@Entity
@SequenceGenerator(
      name="biereIdseq",
      initialValue=1,
      allocationSize=100)
public class Biere {

   @Id
   @GeneratedValue(
         strategy=GenerationType.SEQUENCE,
         generator="biereIdseq")
   private Long id;
…
}
```

Identifiant composite

Utiliser une classe contenant les champs

```
public class PersonnePK {
  private String firstName;
  private String lastName;
  public PersonnePK( String f, String l) { ... }
  public boolean equals(Object other) { ... }
  public int hash() { ... }
}
```

Déclarer la classe et indiquant les champs id.

```
@Entity
@IdClass(PersonnePK.class)
public class Personne {
  @Id private String firstName;
  @Id private String lastName;
}
```

Mapping avec la base de données (1/2)

Chaque classe de bean entité est associée à une table

- Par défaut, le nom de la table est le nom de la classe
- Sauf si annotation @Table(name="Participant")

- Deux modes de définition des colonnes des tables (donc des attributs persistants).
  – Mode "field" → annote les attributs
  – Mode "property" → annote les méthodes `getter/setter`

  Il faut choisir l'un des 2 modes et utiliser le même partout dans vos classes (Developer eXperience)

- Le nom de la colonne est le nom de l'attribut
  Sauf si annotation @Column(name="lastName")
  Attention aux mots réservés SQL (Select From Where Order By Having …)

Mapping avec la base de données (2/2)

@Enumerated : définit une colonne avec des valeurs énumérée. EnumType :

- ORDINAL : la position dans l'enum,
- STRING : le nom de l'element

```
public enum UserType {
        STUDENT, TEACHER, SYSADMIN};
…
@Enumerated(value=EnumType.STRING)
protected UserType userType;
```

@Temporal : datesTemporal Type :

- DATE (java.sql.Date),
- TIME (java.sql.Time),
- TIMESTAMP (java.sql.Timestamp)

```
@Temporal(TemporalType.DATE)
protected java.util.Date creationDate;
```

Entity Manager

- Assure la correspondance entre les objets Java et les tables relationnelles
  - point d'entrée principal dans le service de persistance
  - permet de faire persister les beans
  - permet d'exécuter des requêtes
- Accessible via une injection de dépendance
  - attribut de type javax.persistence.EntityManager
  - annoté par @PersistenceContext
- L'API :

*void **persist**(Object o)*
*void **remove**(Object o)*

*<T> T **find**(Class<T> aClass, Object o)*
*Query **createQuery**(String query)*

*<T> T **merge**(T t) // pour ratacher l'entity à la transaction*

/!\ S'utilise avec une transaction!

Recherche par clé primaire

```
Book myBook = em.find(Book.class,12);
```

- Retourne null si la clé n'existe pas dans la table
- Emet une IllegalArgumentException si
  - 1er paramètre n'est pas une classe d‘Entity
  - 2nd paramètre ne correspond pas au type de la clé primaire

Mise à jour d’un entity

- Mise à jour d'un entity après recherche dans un session bean

```
Book b = em.find(Book.class, 12);
b.setAuthor("tutu");
b.setTitle("mybook");
```

Modification des champs ➔ Modification en base au moment du commit (UPDATE)

- Mise à jour d'un entity serializé complet

```
public void updateBook(Book b) {
    em.merge(b);
}
```

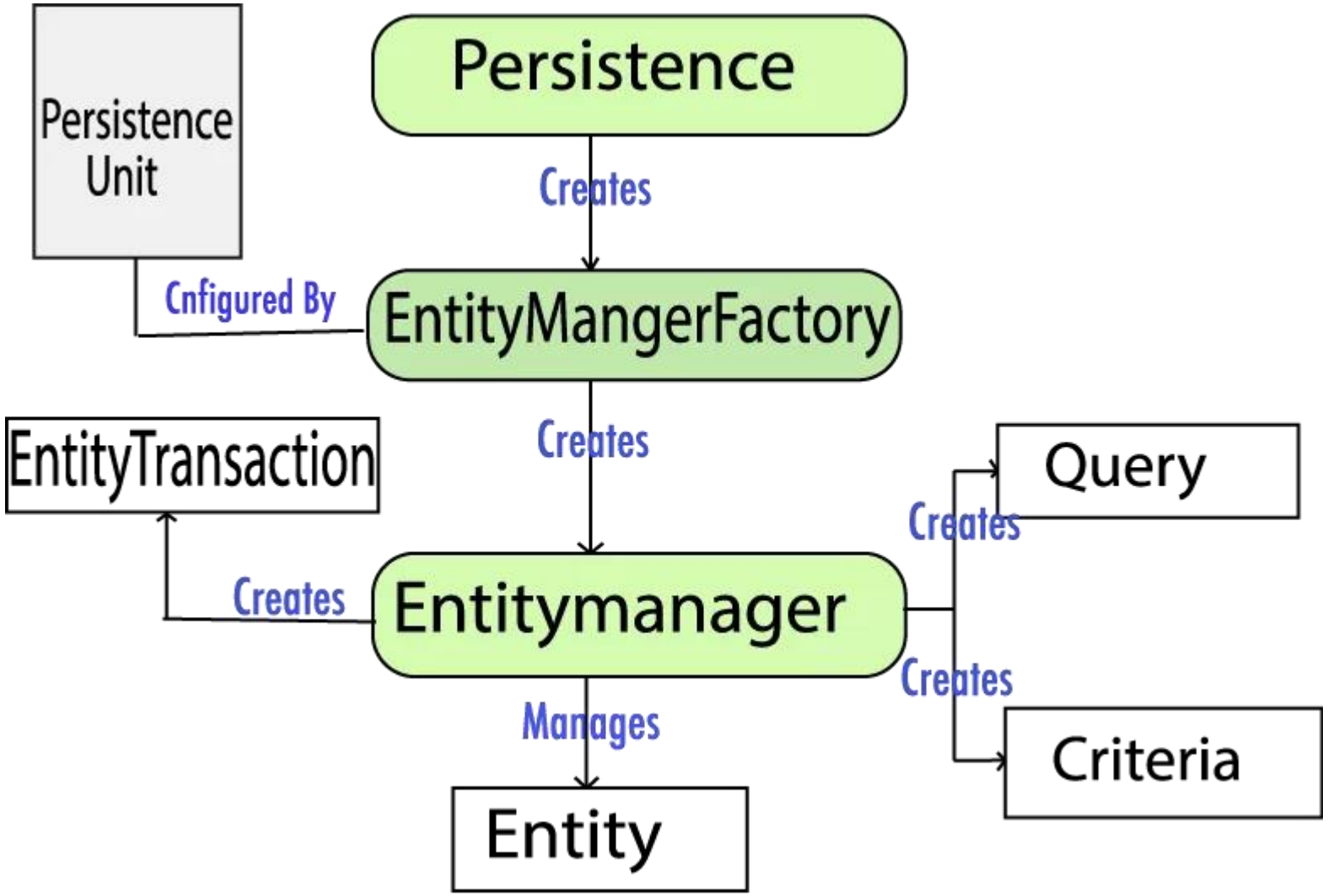
Recherche d’entity par une requête

- Construction d’une requête par une String en JPAQL

```
Query q = em.createQuery("select OBJECT(b) from Book b where b.author = :au");
String nom = "Honore de Balzac";
q.setParameter("au",nom);
List<Book> list = (List<Book>) q.getResultList();
```

- JPAQL manipule les entity et les fields (et non les tables et les colonnes=
- Paramètres nommés (préfixés par :) pour configurer la requête
- Méthode getSingleResult() pour récupérer un résultat
  - uniqueNonUniqueResultException en cas de non unicité
- Possibilité de créer des requêtes en SQL (table/colonne) directement.

API du système de persistance

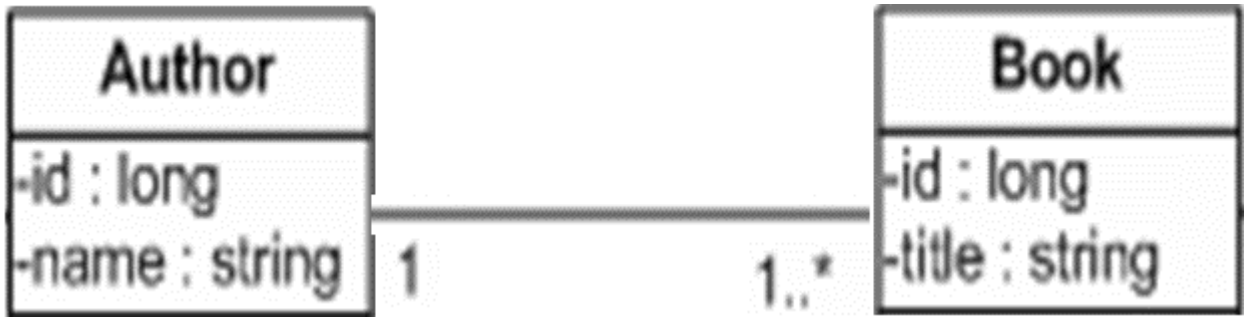


Les relations entre entity : La théorie

- Utilisation d'une annotation pour caractériser la relation entre 2 entity
  - 1-1 @OneToOne
  - 1-N @OneToMany
  - N-M @ManyToMany
  - Unidirectionnel ou bidirectionnel

    Si bidirectionnel il faut mettre les annotations des 2 cotés
- Definition du mapping par des annotations
  - @JoinColumn(name="FK_COL_NAME")
  - @JoinTable(name="JOIN_TABLE_NAME")
  - Si bidirectionnel le mapping est défini d'un côté seulement. De l'autre côté l'annotation @MappedBy(name="fieldName") indique ou trouver le mapping

Les relations entre entity : Un exemple 1-N




```
@Entity
public class Author {

 @Id private long id;
  private String name;

  @OneToMany(mappedBy="author")
  private Collection<Book> books;
}
```

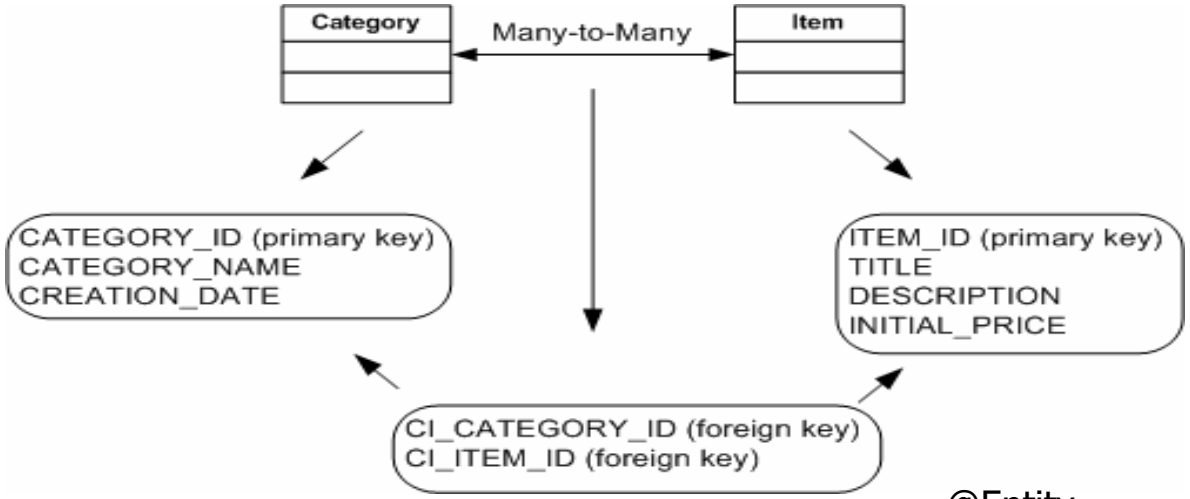
```
@Entity
public class Book {

  @Id private long id;
  private String name;

  @ManyToOne()
  @JoinColumn(name="author_fk")
  private Author author;
}
```

La table ‘book’ contient une colonne ‘author_fk’ qui point sur la colonne id de la table ‘author’.

Les relations entre entity : Un exemple M-N




```
@Entity
public class Category {
   @Id @Column(name="CATEGORY_ID")
   private long categoryId;
   @ManyToMany
   @JoinTable(name="CATEGORIES_ITEMS",
      joinColumns=@JoinColumn(
               name="CI_CATEGORY_ID",
               referencedColumnName="CATEGORY_ID"),
      inverseJoinColumns=@JoinColumn(
                  name="CI_ITEM_ID",
                  referencedColumnName="ITEM_ID"))
    protected Set<Item> items;
}
```

```
@Entity
public class Item {
      @Id @Column(name="ITEM_ID")
      protected long itemId;

      @ManyToMany(mappedBy="items")
      private Collection<Category> categories;
}
```

Les relations entre entity : caractérisation de la relation

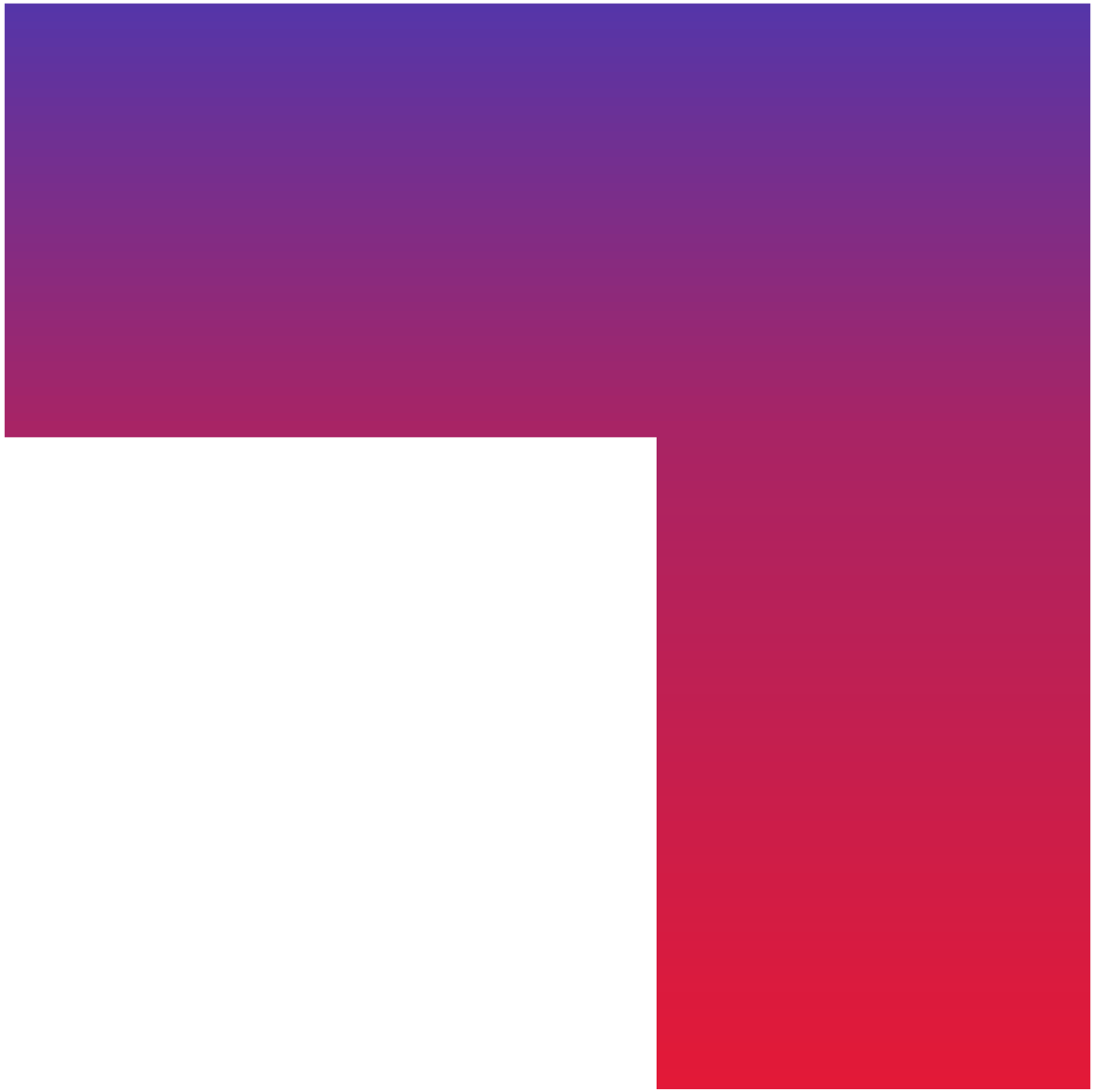
- Mode de chargement d'une relation
  - Attribut Fetch sur l'annotation d'une relation
  - EAGER : chargement immédiat de la ou les entity référencés
  - LAZY : chargement à la demande de la ou les entity référencés (besoin d'une Transaction)

➔ Choisir LAZY par défaut pour éviter les problèmes de performances.
Puis changer pour EAGER selon les cas d'usage
Utiliser aussi les EntityGraph

- Persistance ou suppression en cascade pour une relation
  - Attribut Cascade sur l'annotation d'une relation
  - CascadeType.ALL, CascadeType.PERSIST ...

@OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)

Spring Data JPA



Sébastien Chassande-Barrioz
sebastien.chassande-barrioz@cgi.com

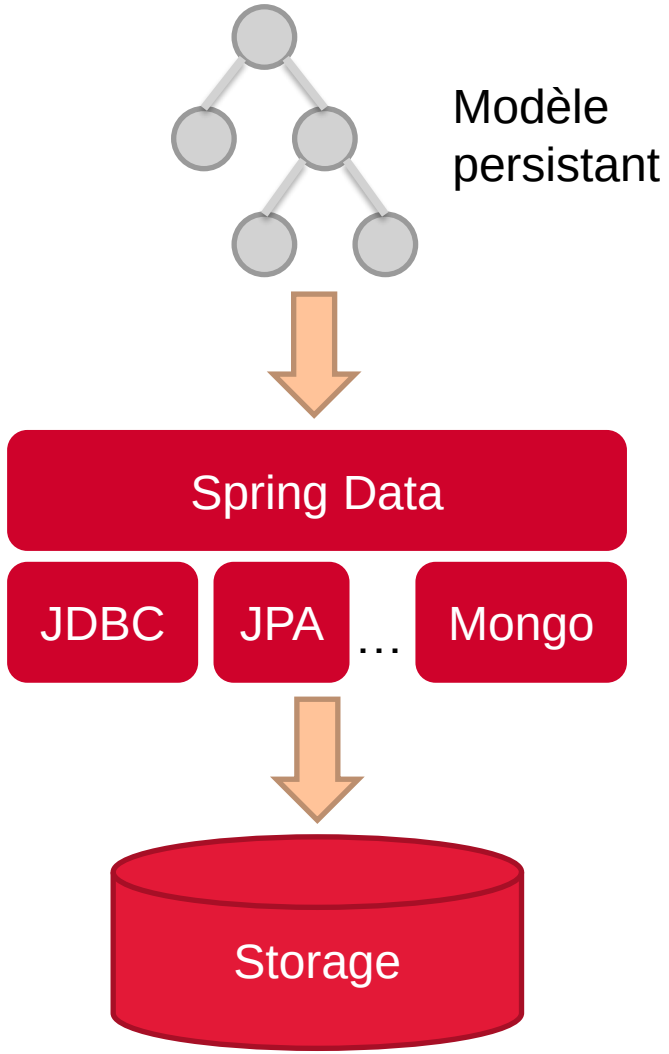
CGI

Les relations entre entity : caractérisation de la relation

- Spring Data core = un standard Spring pour la persistance d'objet métier sur un support de stockage
- Déclinaison : Jdbc, JPA, Mongo DB, Redis, Solr, Hadoop, REST, NEO4J, Cassandra …

- **/!** Spring data n'est pas un ORM mais définit une API encore plus générique que JPA.

- **Mais** *Spring Data Jdbc* est un pseudo ORM



Déclarer un repository pour son entity

Utiliser une classe contenant les champs

```
@Entity
public class Book {

  @Id private String title;
  private String genre;
  …
}
```

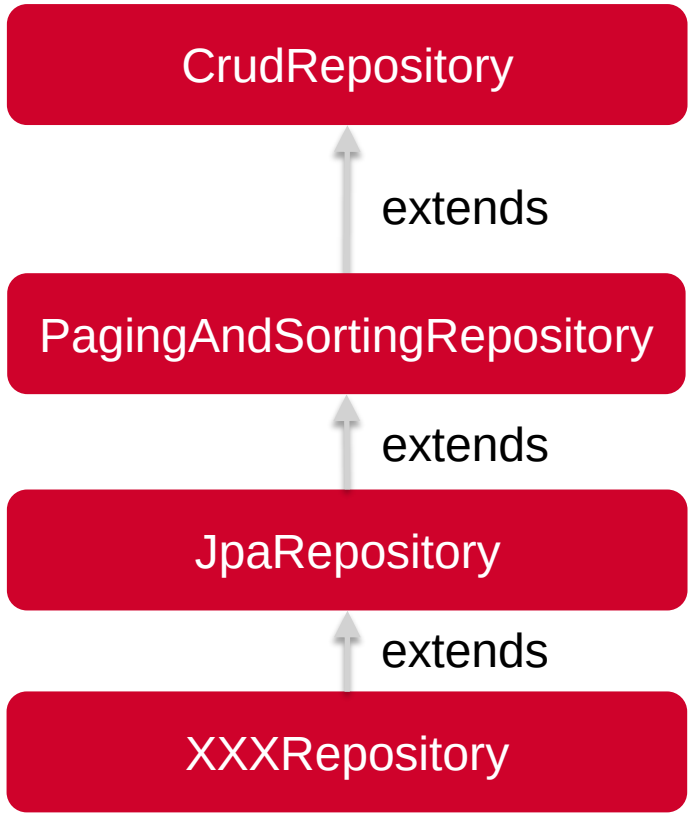
```
public interface BookRepository
                 extends JpaRepository<Book, String> {

  //déclaration des requetes spécifiques ici

}
```

C’est quoi un Repository Spring data ?

- Repository = Un bean gérant la persistance d’un POJO.
- Principe : Ecrire une interface et Spring fournit une implémentation



- **save**
- **existsById**
- **delete**
- **findById**
- **findAll**
- **count**

- **findAll**(Sort sort)
- Page<T> **findAll**(Pageable pageable)

- **deleteAllInBatch, deleteInBatch**
- **saveAll**
- **flush**

- **Custom finder**
- **Custom update**
- **Custom delete**

Définition de query par annotation

- Dans le Repository
- @Query, @Param
- JPQL (Java Persistence Query Language)
- Validation au démarrage

```
public interface TaskRepository
                 extends JpaRepository<Task, Long> {

  @Query("select t from Task t where t.number IN (:numbers)")
  List<Task> findByNumberIn(
       @Param("numbers") Collection<String> numbers);

}
```

Comment configurer ?

Par annotation :

```
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@EnableJpaRepositories
class Config {}
```

Par XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:jpa="http://www.springframework.org/schema/data/jpa"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/data/jpa
            http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">
    <jpa:repositories base-package="com.acme.repositories"/>
</beans>
```