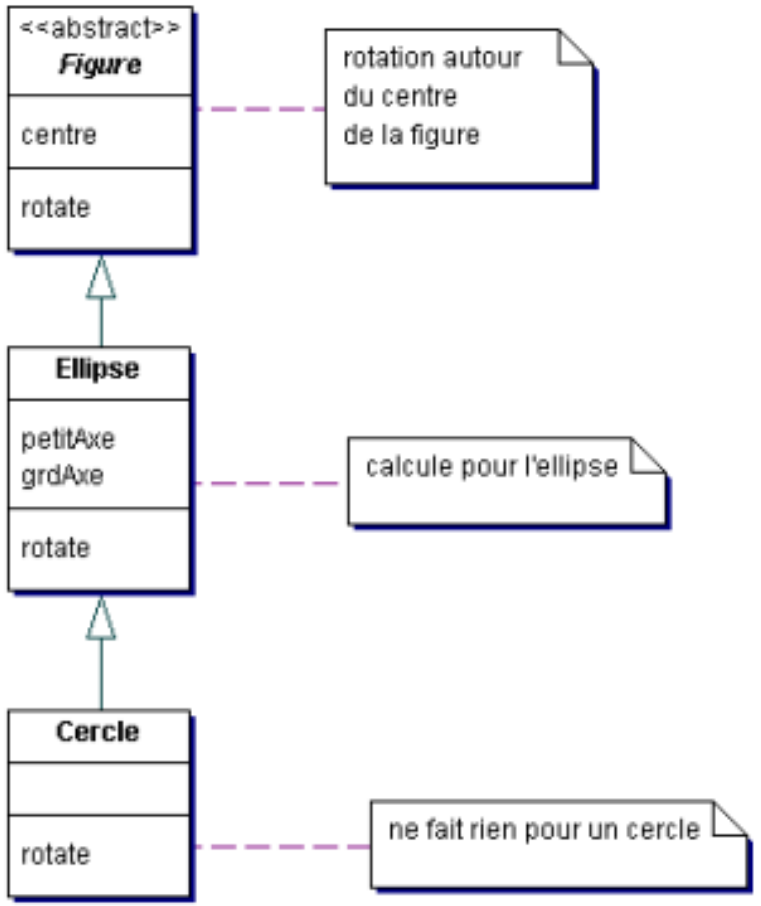
# Mapping de l’héritage

# Plan

- Le  problème
- Relations polymorphes
- Approches du mapping de l’héritage
  - Une table par classe concrète
  - Une table par hiérarchie
  - Une table par classe

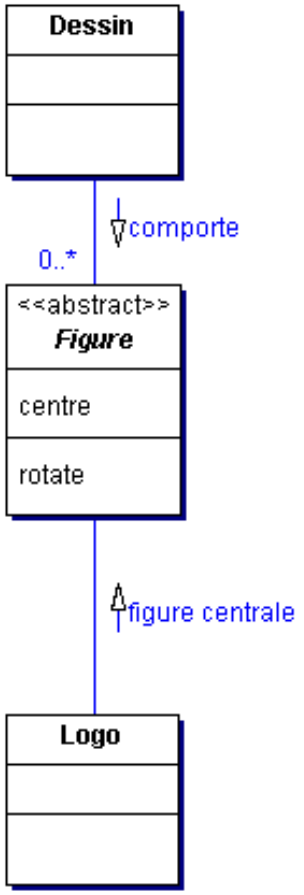
# Le problème

- En relationnel, il n'existe pas de notion d'héritage
  - Structurel
  - Comportemental
- Voir l'héritage comme une forme d'association entre tables représentant les classes.



# Relations polymorphes

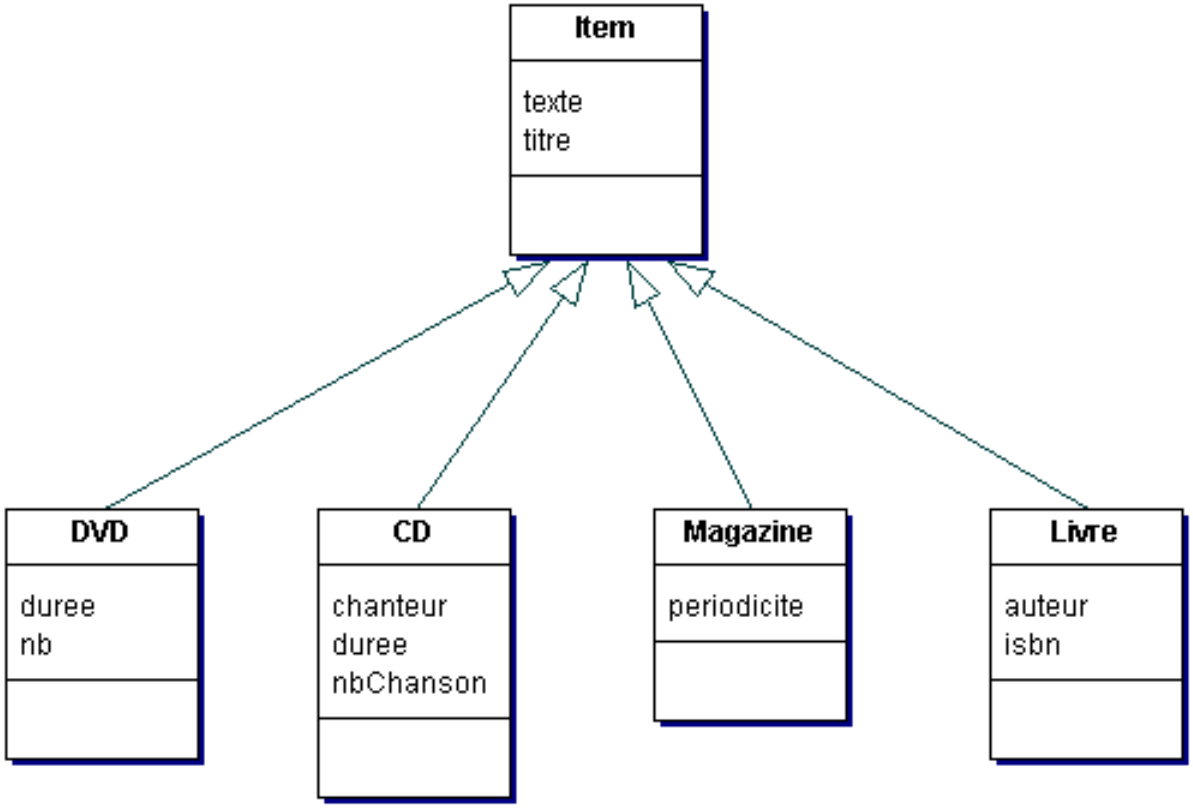
- La liste des figures d'un dessin est polymorphe.
- La figure centrale du Logo peut être de n'importe quel type. L'usage d'une clé étrangère ne suffira pas dans certaines approches.



# Trois approches

- Une table par classe concrète
  - Supprime les relations d'héritage et de polymorphisme du modèle relationnel
- Une table par hiérarchie de classes
  - Permet le polymorphisme par emploi d'une colonne discriminante
- Une table par classe
  - Représente la relation 'est-un ' en employant une clé étrangère.

# Conception relationnelle d'une hiérarchie de classes



# Une table par classe concrète
## exemple de table

- Chaque table de classe concrète reprend les attributs de la superclasse souvent abstraite(ici Item).

### Table DvD

| id | Titre | Texte | Durée |
|---|---|---|---|
| 1 | Troie | ... | 2h20 |
| 2 | Paris | ... | 1h40 |

### Table CD

| id | Titre | Texte | Chanteur | Nb |
|---|---|---|---|---|
| 1 | Live U2 | ... | U2 | 22 |
| 2 | No rest | .. | Rivers | 10 |

# Une table par classe concrète
## avantages / inconvénients

- Avantages :
  - Pour chaque instance, toutes ses données sont accessibles en une seule table.
  - Facilité d'accès et de reporting.
- Inconvénients :
  - Toute modification structurelle des classes abstraites impacte toutes les tables.
  - Contraintes d'intégrité générales plus difficiles à implémenter.
  - Les associations polymorphiques (association avec la superclasse) posent problème.
  - Une requête polymorphique sur la superclasse nécessite autant de requêtes que de tables / classes concrètes.

# Une table par classe concrète
## exemple fichier de mapping avec union

```
<hibernate-mapping  package="com.plb.etechno.j12.exemple.metier">
        <class name="Item">
                    <id  name="id" column="id" access="field">
                          <generator class="native"/>
                     </id>
                    <property name="titre" column="titre"/>
                    <property name="texte" column="texte"/>
                    <set name="exemplaires"   lazy=" false" >
                             <key column="IDItem"/>
                             <one-to-many class="Exemplaire"/>
                    </set>
              <union-subclass name="DVD" table="TDvD">
                <property name="duree" column="duree"/>
                <property name="nbDvd" column="nb"/>
              </union-subclass>
              <union-subclass name="CD" table="TCd">
                <property name="chanteur" column="chant"/>
                <property name="nbDvd" column="nb"/>
              </union-subclass>
        </class>
</hibernate-mapping>
```

Partie Item

Partie DvD

Partie CD

# Une table par classe concrète

Il suffit en tout et pour tout d'utiliser l'annotation *@Inheritance* sur la classe de base en indiquant la stratégie utilisée:

***@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)***

# Une table par classe concrète

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Item implements java.io.Serializable{
...
}
@Entity
public class DVD extends Item {
...
}
@Entity
public class CD extends Item {
...
}
```

# Une table pour une hiérarchie
## Exemple de table

- La table contient tous les attributs de la hiérarchie.
- Ajouter un attribut de type si héritage simple ou plusieurs attributs booléens si héritage multiple.

### Table Item

| **id** | titre | texte | duree | nbDvd | chanteur | auteur | isbn | periodic. | **type** |
|---|---|---|---|---|---|---|---|---|---|
| 01 | Troie | Fil. | 2h20 | 2 | null | null | null | null | DvD |
| 02 | Supe | Dis. | 1h04 | null | SuperT. | null | null | null | CD |
| 03 | Tele7 | Je.. | null | null | null | null | null | hebdo | Mag |

# Une table pour une hiérarchie
## Avantages / Inconvénients

- Avantages :
  - Simplicité
  - Toutes les données d'un item sont dans une seule table (facilite l'accès et le reporting).
  - Facile de modifier la structure ou d'ajouter une classe.
- Inconvénients :
  - A chaque ajout d'un attribut dans la hiérarchie, on impacte la table.
  - Perte potentielle de place en BD
  - Les attributs de même nom provenant de plusieurs branches doivent être renommés.
  - Les attributs non partagés ne peuvent faire l'objet d'une contrainte NOT NULL
  - La table peut grossir rapidement

# Une table pour une hiérarchie
## Exemple fichier de mapping

```
<hibernate-mapping  package="com.plb.etechno.j12.exemple.metier">
   <class name="Item" table="TItem" discriminator-value="IT">
      <id  name="id" column="ID"><generator class="native"/></id>
      <discriminator column="discriminator" type="string"/>
      <property name="titre" column="titre" />
      <property name="texte" column="texte" />
      <set name="exemplaires" inverse="true" cascade="all-delete-orphan"  lazy=" false" >
         <key column="IDItem"/>
         <one-to-many class="Exemplaire"/>
      </set>

      <subclass  name="DvD''  discriminator-value="DVD">
            <property name="nbDvd" column="nbdvd"/>
            <property name="duree" column="duree"/>
      </subclass>

      <subclass name="CD''   discriminator-value="CD">
            <property name="chanteur" column="chanteur"/>
      </subclass>

   </class>
</hibernate-mapping>
```

Item

DvD

CD

# Une table pour une hiérarchie
## Balise <subclass>

- Définit les sous-classes dans un fichier de mapping une table pour une hiérarchie.
- Le fichier concerne la classe racine de la hiérarchie.
- Attributs importants :
  - name : nom qualifié de la sous-classe
  - discriminator-value : la valeur identifiant ce sous-type

# Table unique JPA

L'annotation ***@Inheritance*** sur la classe de base précise la stratégie utilisée:

*@Inheritance(strategy=InheritanceType.SINGLE_TABLE)*

L'annotation ***@DiscriminatorColumn*** précise la colonne utilisée pour identifier le type du bean entité:

*@DiscriminatorColumn(name="DISCRIMINATOR", discriminatorType=DiscriminatorType.STRING)*

- le nom de cette colonne est DTYPE par defaut
- son type peut être STRING (par défaut), CHAR ou INTEGER

L'annotation ***@DiscriminatorValue*** précise la valeur donnée au champ pour identifier la classe de bean entité (nom de la classe par défaut si STRING)

# Exemple JPA

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="DISCRIMINATOR",
   discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue("ITEM")
public class Item implements java.io.Serializable{ … }


@Entity @DiscriminatorValue("DVD")
public class DVD extends Item { … }


@Entity @DiscriminatorValue("CD")
public class CD extends Item { … }
```

# Une table par classe
## Exemple de tables

- Autant de tables que de classes mais avec une clé primaire partagée permettant le lien d'héritage.

table Item

| **id** | titre | texte |
|---|---|---|
| 01 | Troie | Film de… |
| 02 | Collatéral | Thriller… |
| 03 | Crisis | Premier… |

table TDvD

| **IdDVD** | duree | nbDvD |
|---|---|---|
| 001 | 2h20 | 2 |
| 002 | 2h | 1 |

table TCD

| **idCD** | chanteur |
|---|---|
| 003 | Supertramp |

# Une table par classe
## Avantages / Inconvénients

- Avantages :
  - Conforme au modèle objet.
  - Facilité de modification structurelle dans la hiérarchie.
  - Modèle relationnel normalisé.
  - On peut placer les contraintes d'intégrité.
- Inconvénients :
  - Nécessite l'accès à plusieurs tables pour retrouver toutes les données d'une instance.
  - Reporting moins aisé nécessitant des vues pour simuler les tables.

# Une table par classe
## Exemple de fichier de mapping

```
<hibernate-mapping  package="com.plb.etechno.j12.exemple.metier">
   <class name="Item" table="TItem" >
      <id  name="id" column="ID"><generator class="native"/></id>
      <property name="titre" column="titre" />
      <property name="texte" column="texte" />
      <set name="exemplaires" inverse="true" cascade="all-delete-orphan"  lazy=" false" >
         <key column="IDItem"/>
         <one-to-many class="Exemplaire"/>
      </set>

      <joined-subclass  name="DvD''  table=« TDVD">
            <key column=« IDDVD »/>
            <property name="nbDvd" column="nbdvd"/>
            <property name="duree" column="duree"/>
      </joined-subclass>

      <joined-subclass name="CD''   table=« TCD">
            <key column=« IDCD »/>
           <property name="chanteur" column="chanteur"/>
      </joined-subclass>

    </class>
</hibernate-mapping>
```

# JPA, Une table par sous-classe

l'annotation @*Inheritance* sur la classe de base précise la stratégie utilisée:

***@Inheritance(strategy=InheritanceType.JOINED)***

# Une table par sous-classe

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Item implements java.io.Serializable{
...
}
@Entity
public class DVD extends Item {
...
}
@Entity
public class CD extends Item {
...
}
```

# Requêtes polymorphes

```
public List<Item> getAllItems(){
  List<Item> ret = new ArrayList<Item>();
  Session session = DBHelper.getFactory().openSession();
  Transaction tx = null;
  try {
     tx = session.beginTransaction();
     Query hqlQuery = session.createQuery("from Item");
     ret = (List<Item>)hqlQuery.list();
     tx.commit();
…
```

CU : Consulter la liste des items
1 - L'utilisateur demande à consulter la liste complète des items
2 - Le système affiche la liste des items
1) Troie nb Ex -> 3
2) Collateral nb Ex -> 0
3) Crisis nb Ex -> 0
3 - L'utilisateur sélectionne un item [le 3 ème]
4 - Le système affiche les détails :
Crisis  chanteur : SuperTramp

# Exercice

- Exercice 4 : Mapping de l’héritage