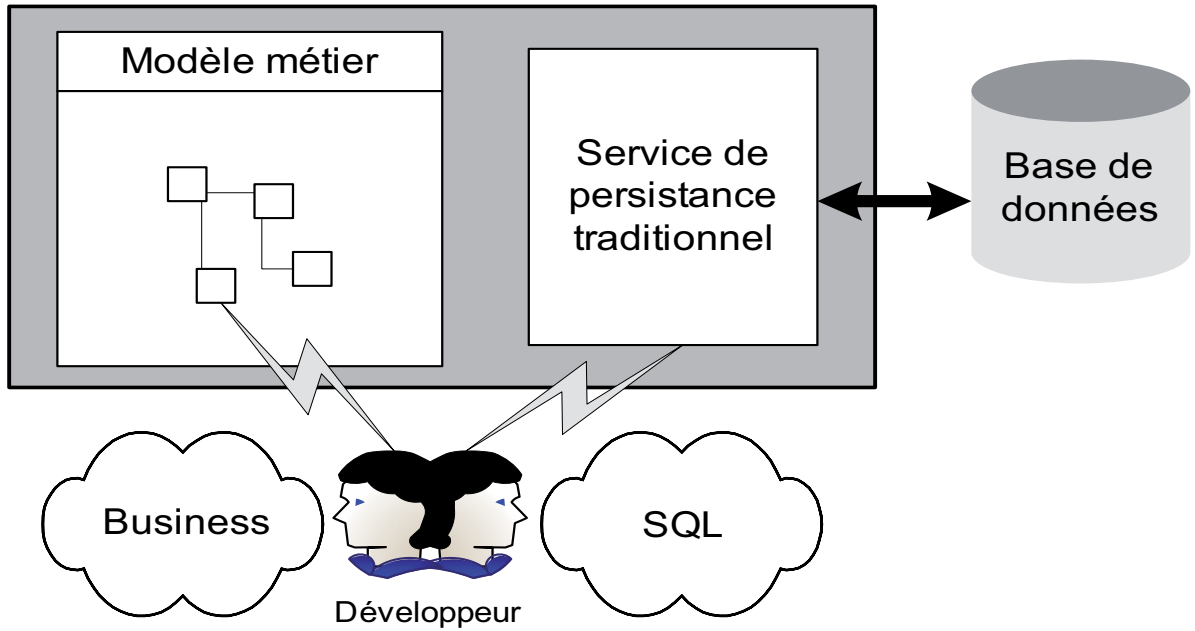
Mapping Objet Relationnel avec JDBC

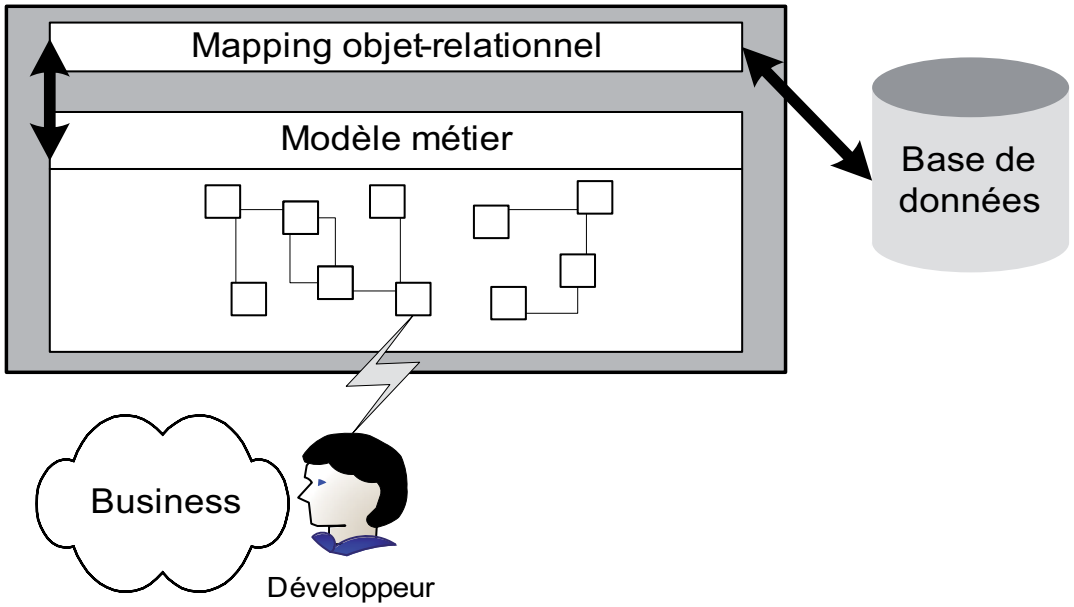
Principes de la persistance des objets



Les applications d'entreprise orientées objet:

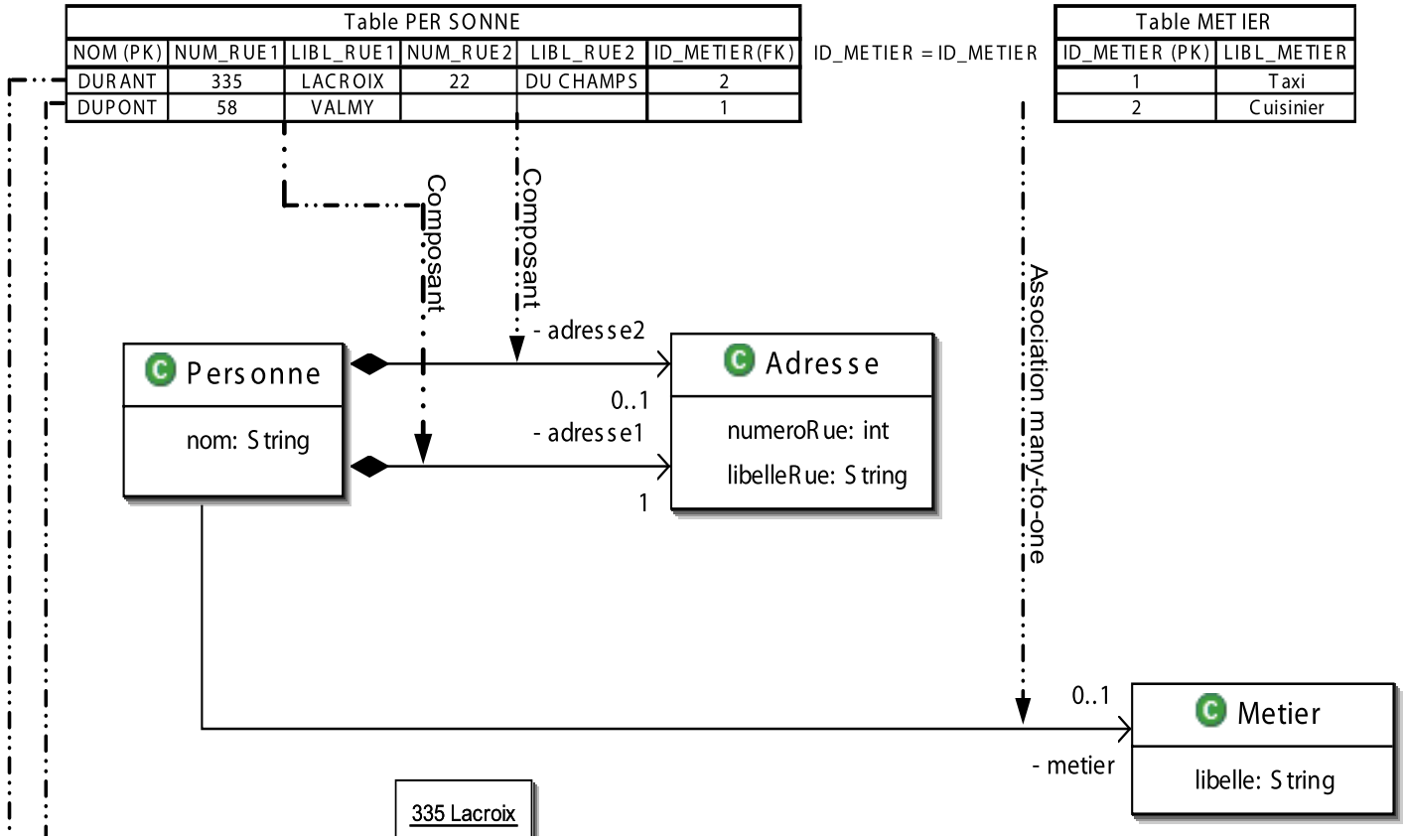
- utilisent **les bases de données relationnelles pour stocker les objets dans des lignes de tables,**
- les **propriétés des objets** étant représentées par **les colonnes des tables.**
- Les **relations entre les objets** sont représentées par **les clés étrangères.**

La couche de persistance



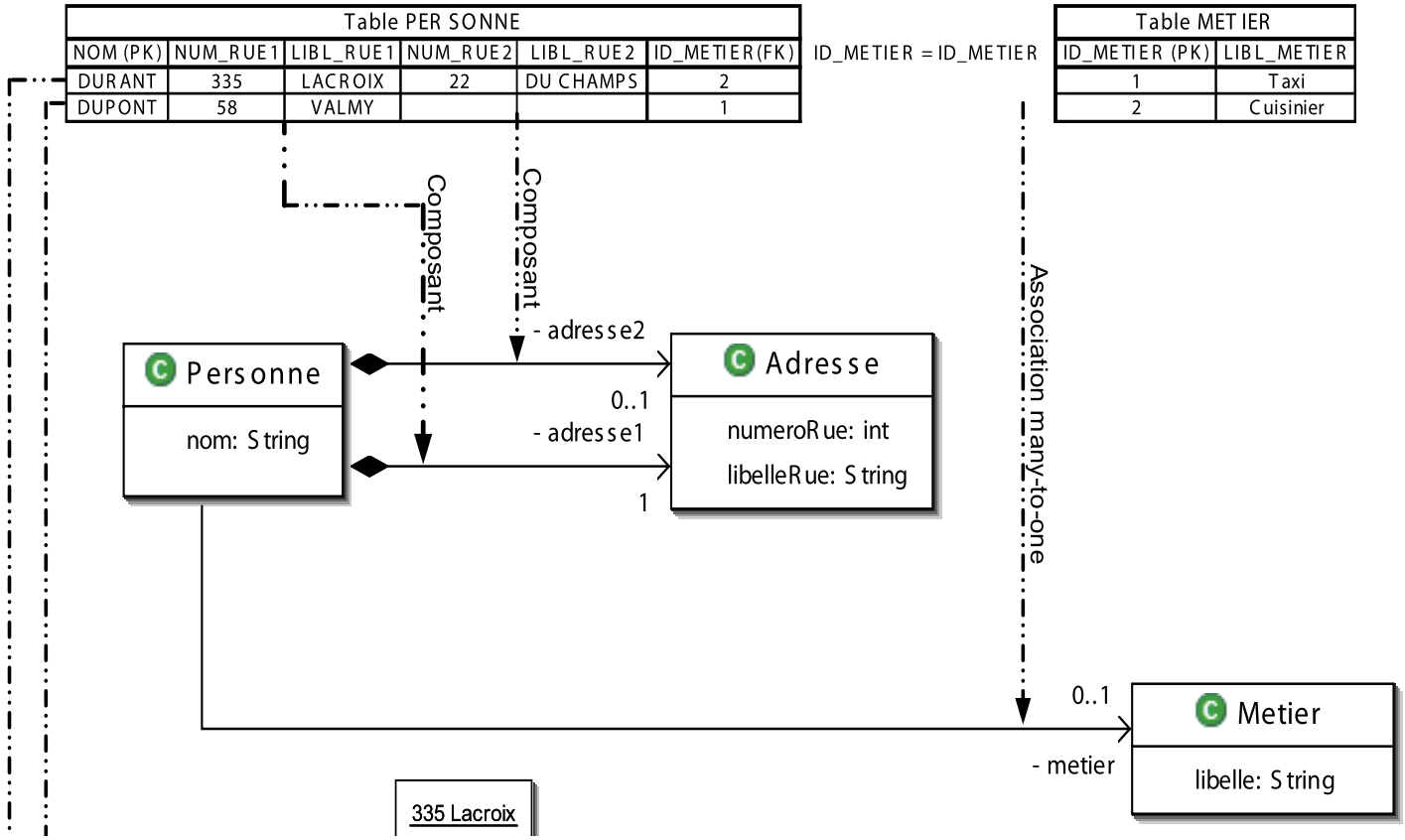
Le développeur utilise **la couche de persistance** comme **un service rendant abstraite la représentation relationnelle** indispensable au stockage final de ses objets.

Le mapping objet-relationnel



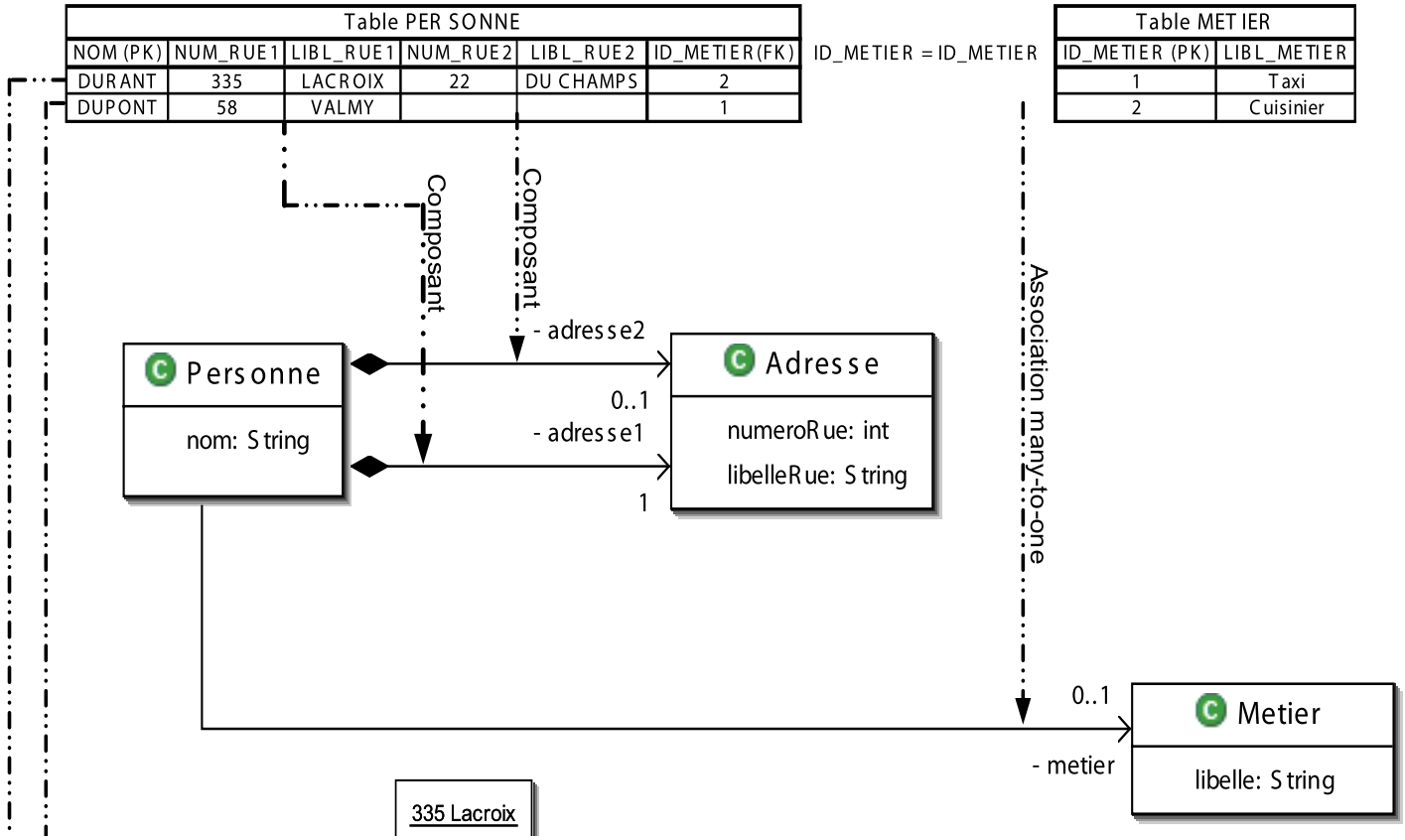
Le principe du **mapping objet-relationnel** consiste à décrire une **correspondance entre un schéma de base de données et un modèle de classes.**

Le mapping objet-relationnel



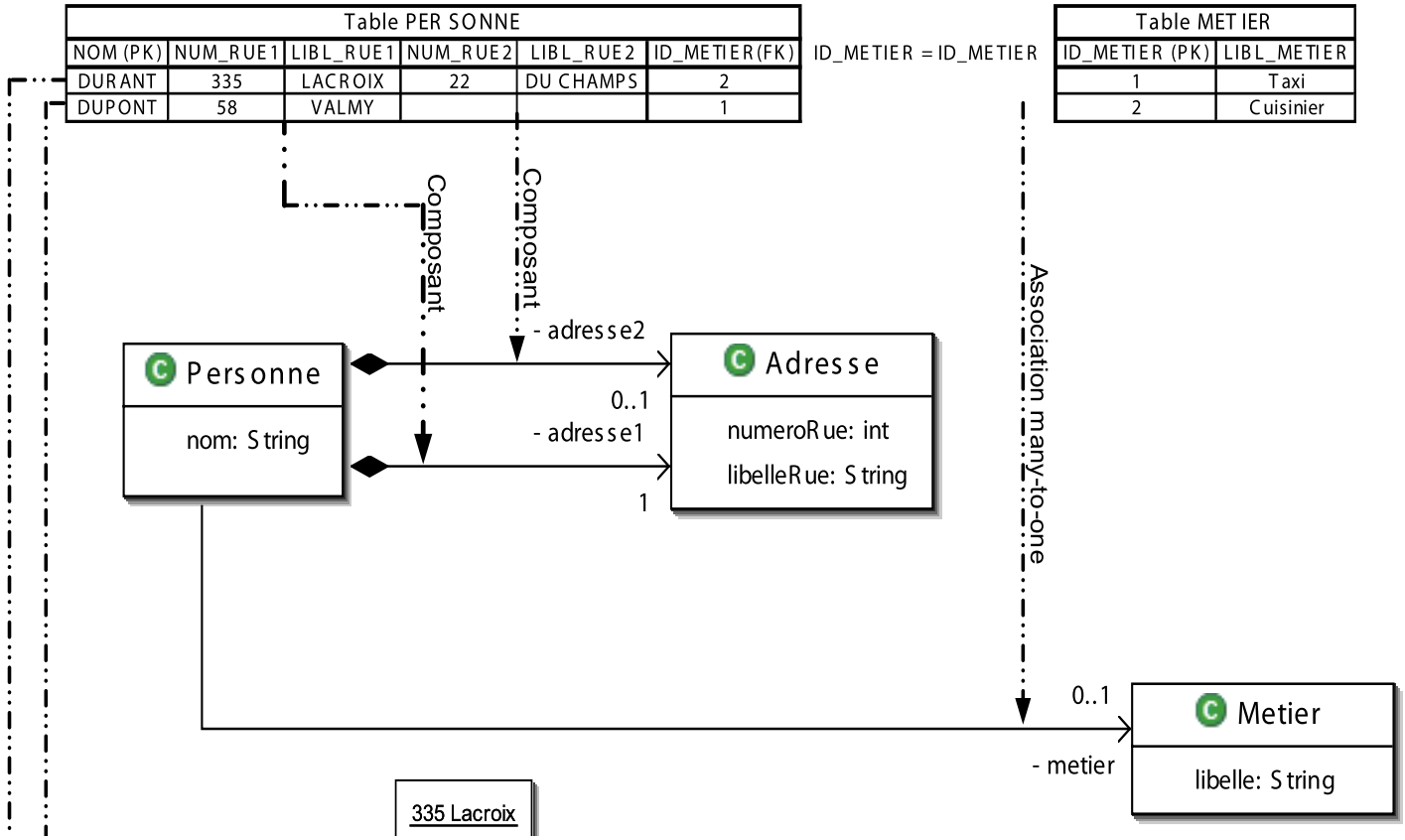
La correspondance ne se limite pas à celle entre la structure de la base de données et le modèle de classes mais concerne aussi celle entre **les instances de classes et les enregistrements des tables**,

Le mapping objet-relationnel



une **seule table** peut reprendre **plusieurs classes**, comme la classe **Adresse**.

Le mapping objet-relationnel



- Les colonnes représentent les propriétés d'une classe,
- les liens relationnels entre deux tables (**clé étrangère**) forment les **associations dans le modèle objet.**
- Les **deux tables sont liées** par la colonne **ID_METIER (clé étrangère)**

Mapping d'objets légers

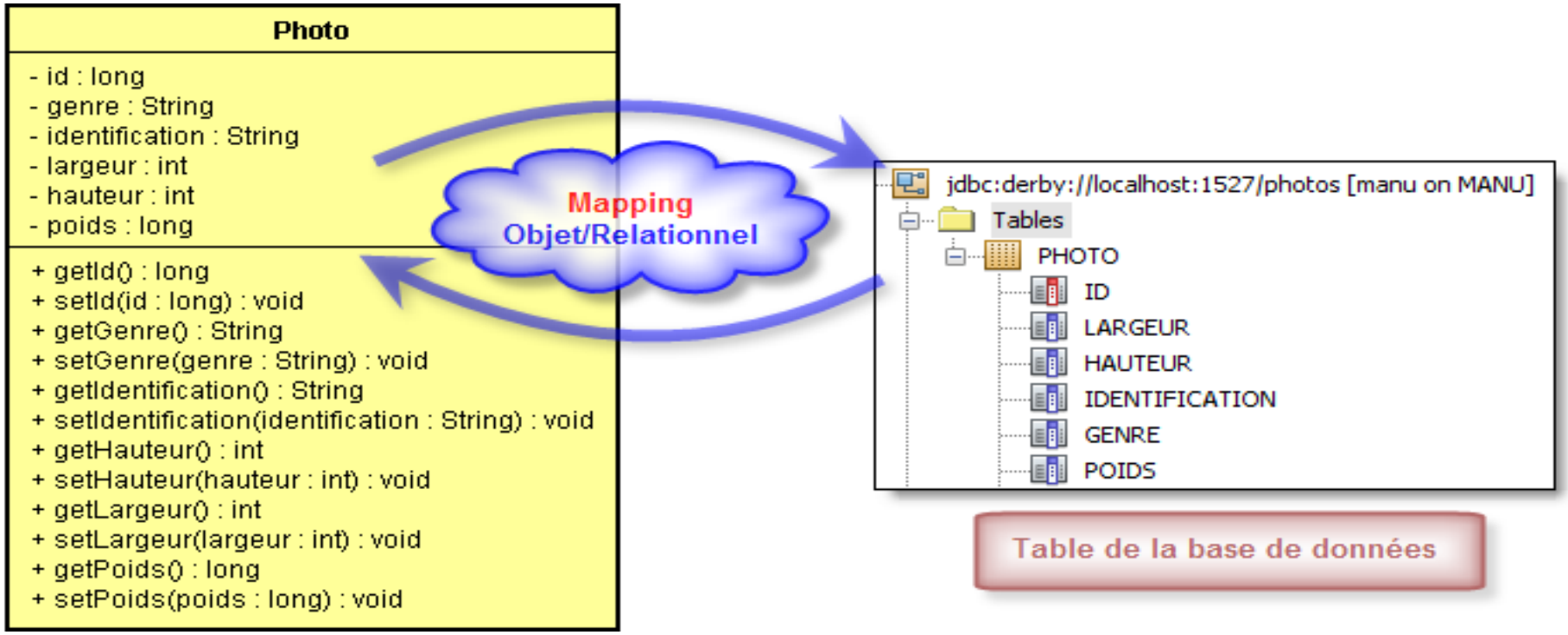
- Les entités sont modélisées comme des **classes mappées manuellement aux tables du modèle relationnel.**

- Le code manuel SQL/JDBC est caché de la logique métier par :
  - des design patterns courants, tel **DAO (Data Access Object)**
  - et l'externalisation des requêtes dans des fichiers XML.

- Cette approche largement répandue est adaptée aux applications disposant de peu d'entités.

- Dans ce type de projet, les procédures stockées peuvent aussi être utilisées.

Mapping Objet Relationnel avec JDBC



Pour chaque objet, il faut implémenter:

- Les accesseurs ***getters*** et ***setters***
- Les méthodes ***equals*** et ***hashcode*** *:* Les méthodes equals et hashcode permettent d'implémenter les règles d'égalité de l'objet. Il s'agit de spécifier les conditions permettant d'affirmer que **deux instances doivent être considérées comme identiques ou non.**

Mapping Objet Relationnel avec JDBC

```
public class Team implements Serializable{
  private Long id;❶
  private String name;❷
  private int nbWon;❷
  private int nbLost;❷
  private int nbPlayed;❷
  private Coach coach;❸
  private Set<Player> players = new HashSet<Player>();❹
```

Si vous **surchargez equals()**:

- il est **indispensable de surcharger aussi hashcode**() afin de respecter le contrat d'Object.
- Si vous ne le faites pas, vos objets ne pourront **être stables en cas d'utilisation de collections de type HashSet, HashTable ou HashMap.**

Mapping Objet Relationnel avec JDBC

```
try {
  // étape 1: récupération de la connexion
  con = DriverManager.getConnection("jdbc:hsqldb:test","sa","");
  // étape 2: le PreparedStatement
  PreparedStatement createTeamStmt;
  String s = "INSERT INTO TEAM VALUES (?, ?, ?, ?, ?)";
  createTeamStmt = con.prepareStatement(s);
  createTeamStmt.setInt(1, myTeam.getId());
  createTeamStmt.setString(2, myTeam.getName());
  createTeamStmt.setInt(3, myTeam.getNbWon());
  createTeamStmt.setInt(4, myTeam.getNbLost());
  createTeamStmt.setInt(5, myTeam.getNbPlayed());
  // étape 3: exécution de la requête
  createTeamStmt.executeUpdate();
  // étape 4: fermeture du statement
  createTeamStmt.close();
  con.commit();
} catch (SQLException ex) {
  if (con != null) {
    try {
      con.rollback();
    } catch (SQLException inEx) {
      throw new Error("Rollback failure", inEx);
    }
  }
  throw ex;
} finally {
  if (con != null) {
    try {
      con.setAutoCommit(true);
      // étape 5: fermeture de la connexion
      con.close();
```

- **Exercice sur le Mapping Objet Relationnel avec JDBC**