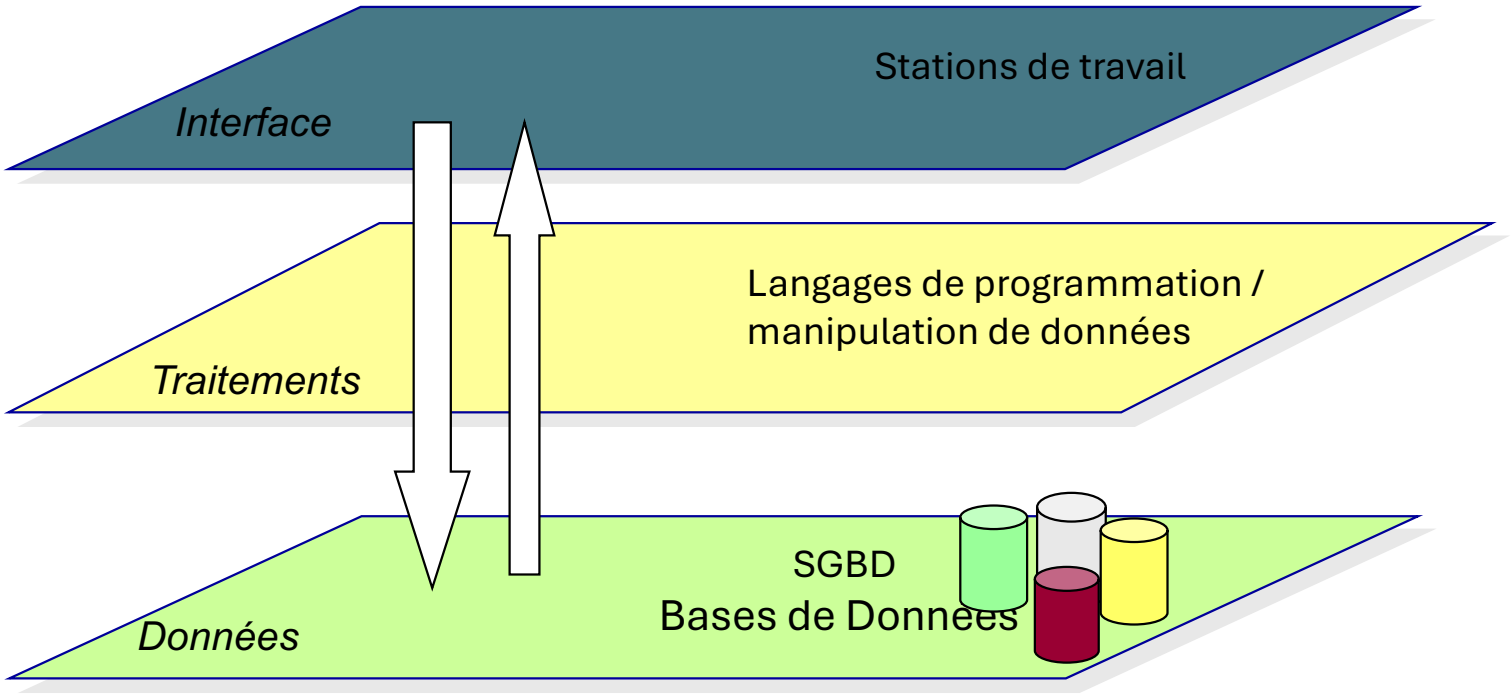
Développement d'applications Java : POO, Hibernate et JavaFX

S4 - ORM et Framework Hibernate

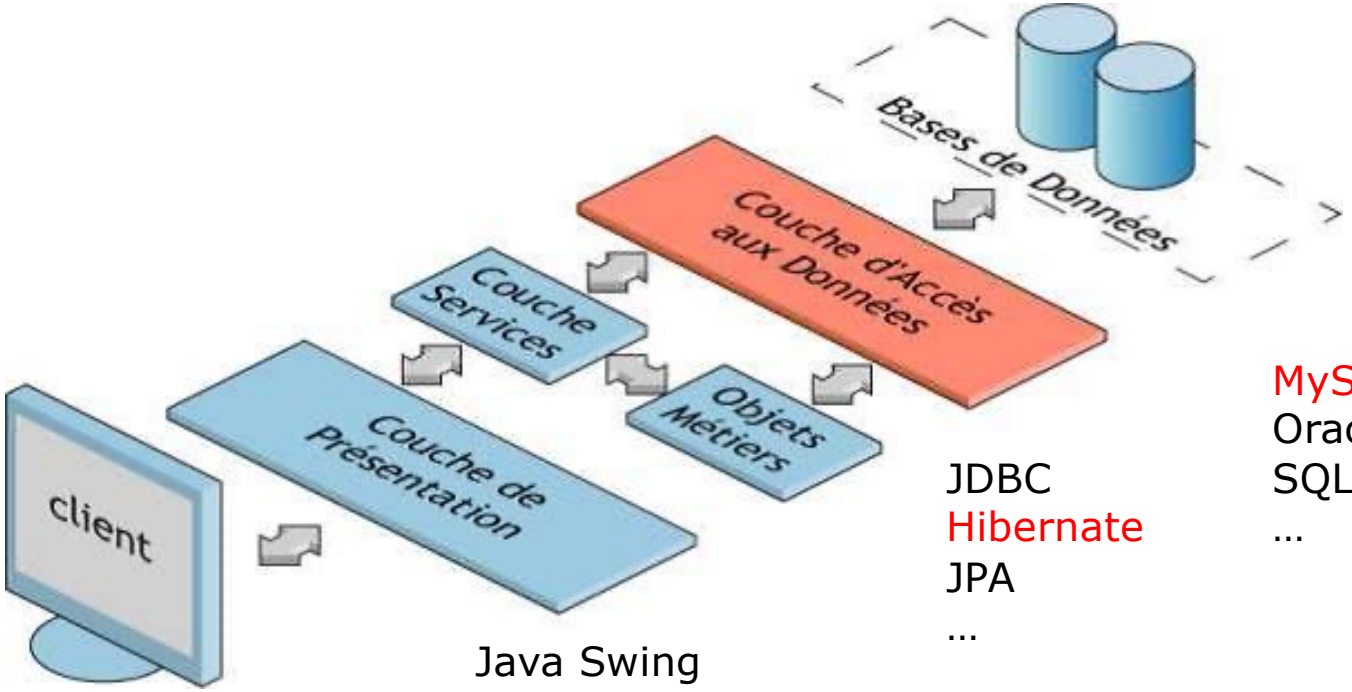
Objectifs de la séance

- Comprendre les structures des bases de données
- Manipuler les données via le Framework Hibernate via JAVA.

Les trois dimensions d'une application



Développement en couche



Java Swing
JAVA FX
JSP / Servlet
Struts
JSF
…

JDBC
Hibernate
JPA
…

MySQL
Oracle
SQLServer
…

Développement en couche

- Dans cette architecture, nous distinguons :
  - le **système de gestion de base de données** (SGBD), qui stocke les données utilisées par l'application,
  - la **couche d'accès aux données**, en charge de l'accès aux données et de leur manipulation, indépendamment du SGBD choisi,
  - la **couche métier**, gère la logique de l'application,
  - la **couche service** : correspondant à la mise en œuvre de l'ensemble des règles de gestion et de la logique applicative,
  - la couche **présentation**, qui s'occupe à la fois d'afficher les données reçues par la couche de services et d'envoyer à la couche de services les informations relatives aux actions de l'utilisateur.

Introduction aux bases de données

Bases de données

« Ensemble structuré de données qui modélisent un univers réel »

Une Base de données est faite pour enregistrer des faits, des opérations au sein d'un organisme (Ecole, administration, banque, université, hôpital, hôtel,..)

Les BD ont une place essentielle dans l'informatique.

Système de Gestion de Base de Données (SGBD)

« Système qui permet de gérer des bases de données partagées par plusieurs utilisateurs simultanément »

Doit permettre de :

- Décrire les données : indépendamment des applications (DATA DEFINITION LANGUAGE)
- Manipuler les données : interroger et mettre à jour les données (DATA MANIPULATION LANGUAGE)
- Contrôler les données : intégrité et confidentialité (DATA CONTROL LANGUAGE)

Relationnel (SQL)
- Oracle
- MySQL
- PostgreSQL
- SQL Server
- MariaDB

NoSQL (Document)
- MongoDB
- CouchDB
- DocumentDB

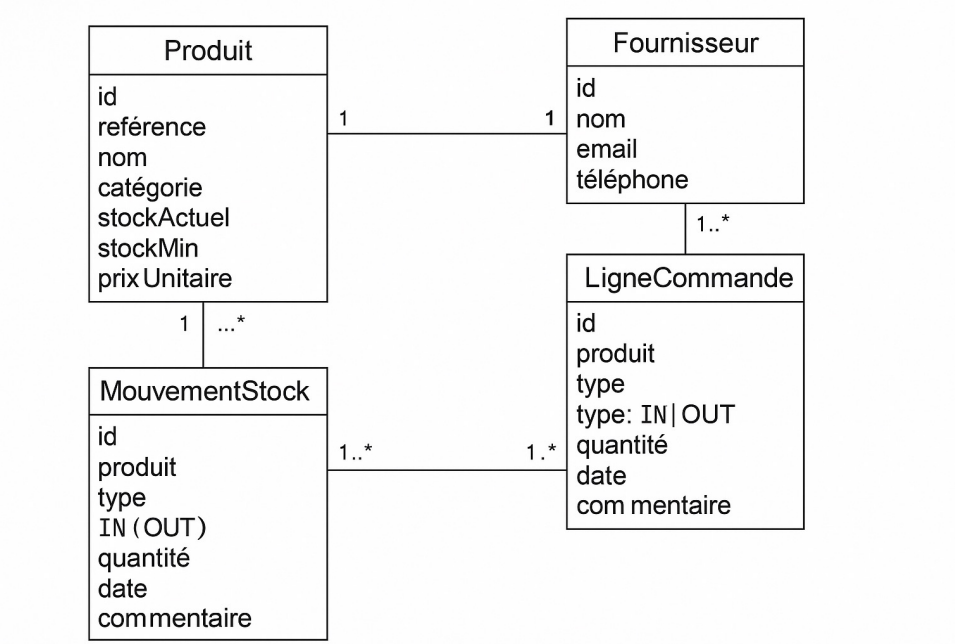
Relationnel (SQL) :
- Données stockées dans des **tables** (lignes/colonnes).
- Schéma fixe : chaque ligne suit la même structure.

NoSQL Document :
- Données stockées sous forme de **documents JSON-like** (clé/valeur imbriquées).
- Schéma flexible : chaque document peut avoir des champs différents.

Le modèle relationnel

Ref	Nom	Prix	Stock
Z12	DELL	5000	10
Z24	SUMSUNG	7000	5
Z26	SONY	6000	8

Id	Type	Produit	quantité	Date
1	In	Z12	100	12/09/2025
2	Out	Z24	2	20/09/2025



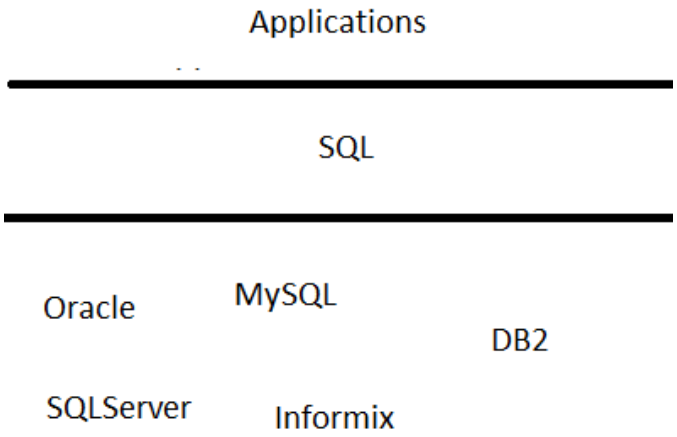
Id	Nom	email	téléphone	produit
1	DELL	contact@dell.com	0546787623	Z12
2	SUMSUNG	contact@sumsung.com	0987654578	Z24
3	SONY	contact@sony.com	0876548932	Z26

Les langages de requête

SQL est un langage **déclaratif** inventé par IBM dans les années 1970 (projet System R).

SQL (Structured Query Language) : SQL est devenu le langage standard pour décrire et manipuler les BDR.

SQL est l'interface logiciel/logiciel entre les applications et les SGDB



Les langages de requête

Les commandes SQL :

- De définition de la base de données :

  CREATE database, DROP database

- De définition des données pour créer au modifier la structure :

  CREATE, DROP, ALTER

- De manipulation des données pour insérer, mettre à jour, supprimer et lire :

  SELECT, INSERT, UPDATE, DELETE

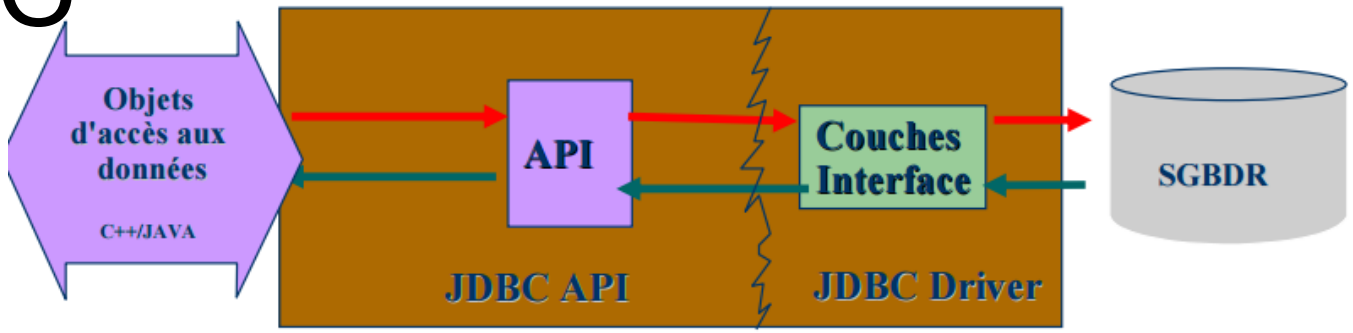
- Transactions

  BEGIN, COMMIT, ROLLBACK

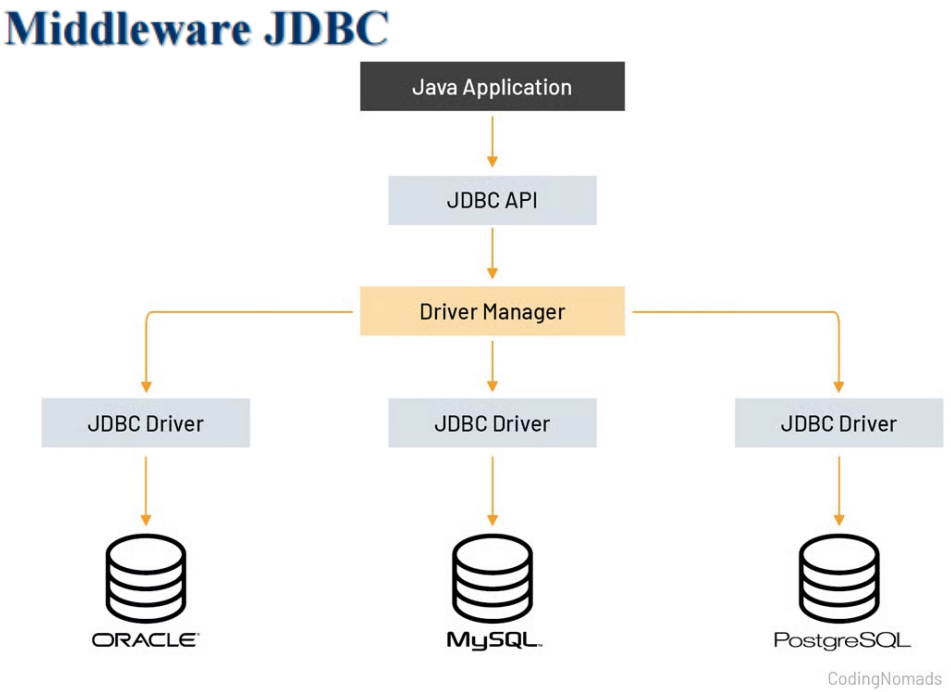
Java : connexion à une base de données

Définition JDBC

Interface de connexion aux bases de données via une API Java ainsi manipuler des requêtes en réseau.



- Java DataBase Connectivity (JDBC)
- Une API Java pour connecter des programmes java à des SGBD.
- Un ensemble de classes et interfaces java.
- JDBC va permettre :
  - D'établir une connexion à la BD.
  - D'envoyer des requêtes SQL.
  - De récupérer les résultats des requêtes.
- Il faut charger un driver pour le SGBD.



Utilisation du JDBC

JDBC API (standard, côté Java)

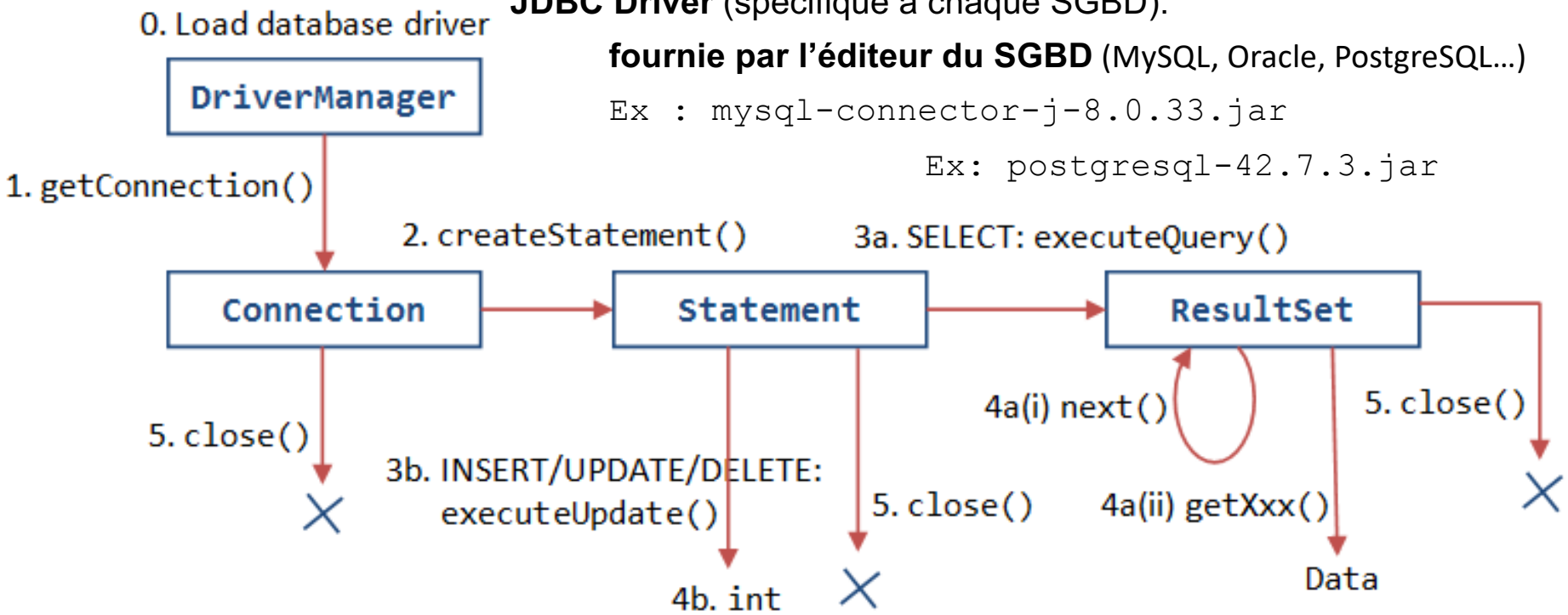
```
Connection con = DriverManager.getConnection(url, user, pwd);
Statement st = con.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM Produit");
```

JDBC Driver (spécifique à chaque SGBD):

fournie par l'éditeur du SGBD (MySQL, Oracle, PostgreSQL…)

`Ex : mysql-connector-j-8.0.33.jar`

`Ex: postgresql-42.7.3.jar`



URL de connexion

- Afin de localiser la base de données sur le serveur de base de données, il est indispensable de spécifier une adresse de connexion sous la forme d'une URL. Ces URL commenceront toutes par "jdbc:".
- Chaque SGBD a un son propre format d'URL.

- Pour MySQL, l'URL est la suivante :

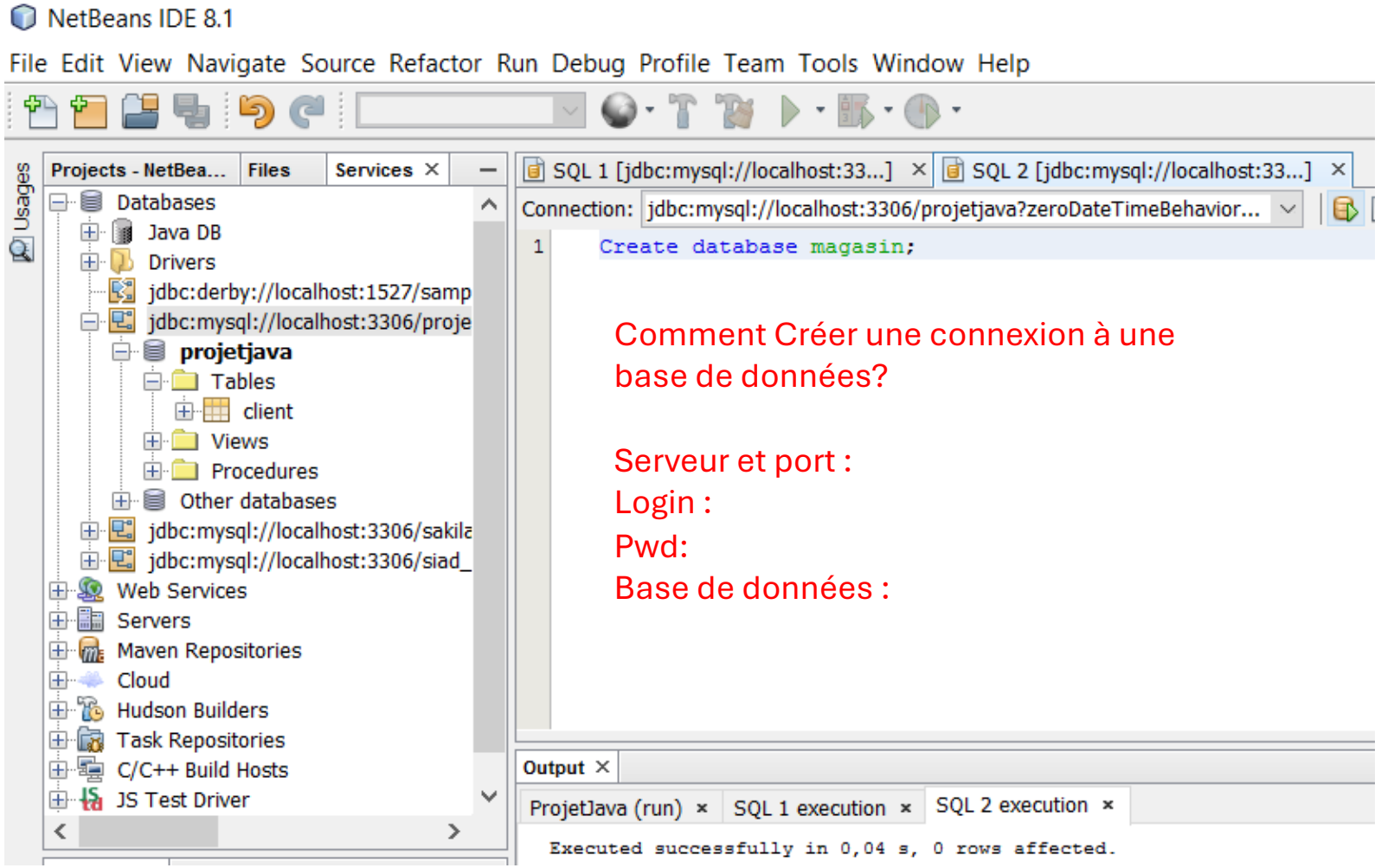
  jdbc:mysql://host:port/database.

  - Host : correspond à l'adresse IP du serveur. (localhost)
  - port : port MySQL ou port par défaut. (3306)
  - Database : le nom de la base de données à laquelle on doit se connecter. (magasin) ➔ (mysql)
  - login de mot de passe (root,) ➔ (root, pwd)

URL de connexion

SGBDR	Nom de la classe du pilote	Format de l'URL de connexion
Oracle DB	oracle.jdbc.OracleDriver	jdbc:oracle:thin:@[host]:[port]:[schema] Ex : jdbc:oracle:thin:@localhost:1521:maBase
MySQL <= 5.1	com.mysql.jdbc.Driver	jdbc:mysql://[host]:[port]/[schema] Ex : jdbc:mysql://localhost:3306/maBase
MySQL >= 8	com.mysql.cj.jdbc.Driver	jdbc:mysql://[host]:[port]/[schema] Ex : jdbc:mysql://localhost:3306/maBase
MariaDB	org.mariadb.jdbc.Driver	jdbc:mariadb://[host]:[port]/[schema] Ex : jdbc:mariadb://localhost:3306/maBase
PosgreSQL	org.postgresql.Driver	jdbc:postgresql://[host]:[port]/[schema] Ex : jdbc:postgresql://localhost:5432/maBase

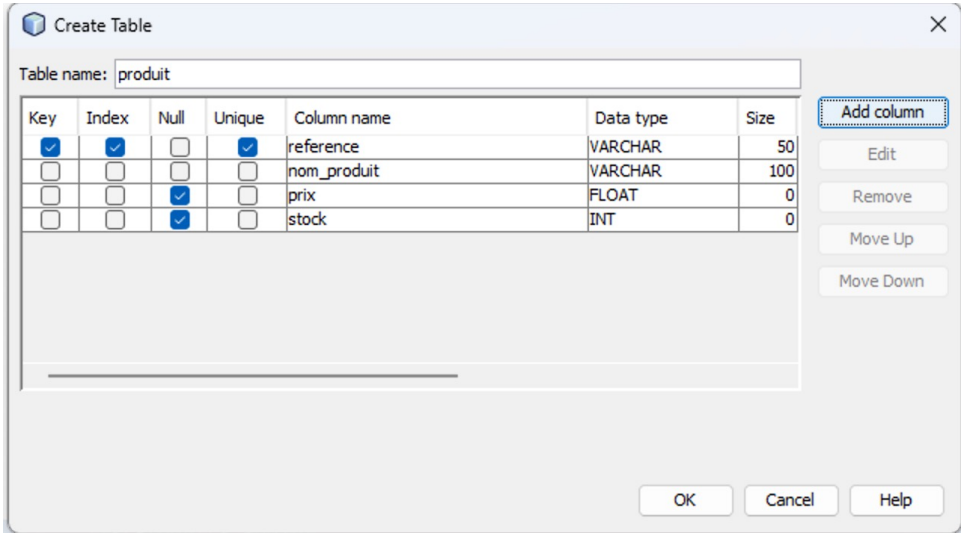
NetBeans service



DB : Exercice

En utilisant l'onglet service de Netbeans

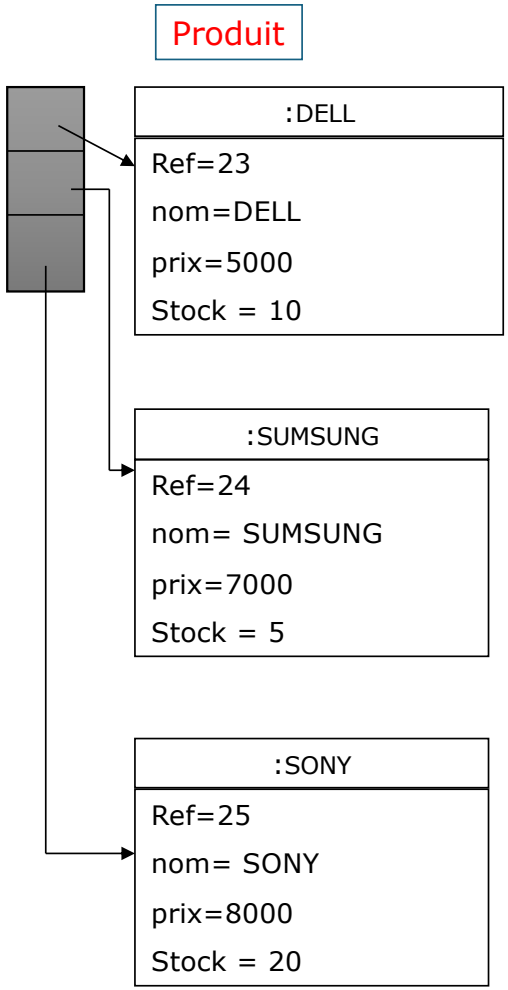
- Créer une base de données « Magasin » create database magasin;
- Créer la table produit (en se référant à votre classe)
- Insérer quelques données dans la table (insert)
- Sélectionner les données (select)



Framework de persistance (Hibernate)

Introduction

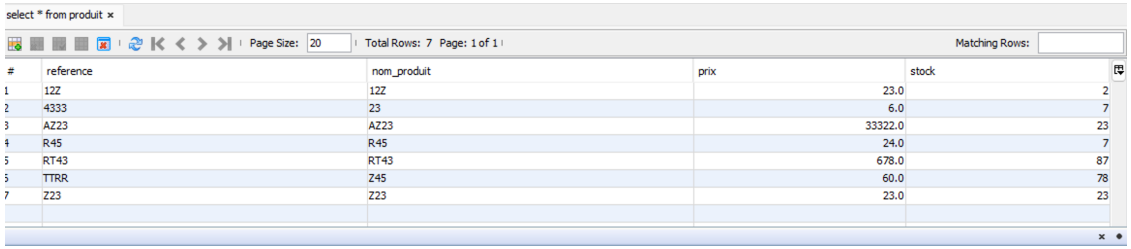
Application orientée objet



Hibernate est un Framework Java de persistance qui permet de faire correspondre des tables de base de données relationnelles avec des objets java.

un programme Java peut manipuler toutes les données en utilisant que des JavaBean, masquant alors totalement la base de données sous-jacente et ses spécificités.

Le Framework assure le remplissage de ces objets et la mise à jour de la base en se basant sur leur contenu.


select * from produit

Page Size: 20 Total Rows: 7 Page: 1 of 1 Matching Rows:

#	reference	nom_produit	prix	stock
1	12Z	12Z	23.0	2
2	4333	23	6.0	7
3	AZ23	AZ23	33322.0	23
4	R45	R45	24.0	7
5	RT43	RT43	678.0	87
6	TTRR	Z45	60.0	78
7	Z23	Z23	23.0	23

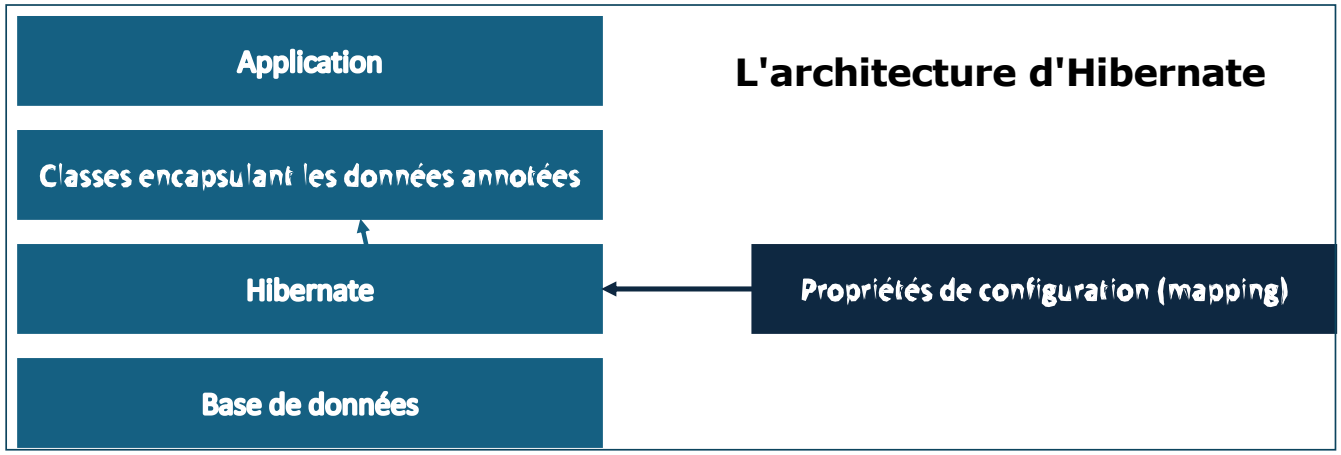
Base de données relationnelle

Introduction

- Hibernate est un Framework de mapping Objet/Relationnel pour applications JAVA.
- Le terme mapping objet/relationnel (ORM) décrit la technique consistant à faire le lien entre la représentation objet des données et sa représentation relationnelle basée sur un schéma SQL.
- Hibernate génère le code SQL :
  - Pas de requête SQL à écrire,
  - Pas d'Objet ResultSet à gérer,
  - Application plus portable. S'adapte à la base de données cible.
- Hibernate s'occupe du transfert des objets Java dans les tables de la base de données
- En plus, il permet de requêter les données et propose des moyens de les récupérer.
- Il peut donc réduire de manière significative le temps de développement qui aurait été autrement perdu dans une manipulation manuelle des données via SQL et JDBC

Mise en œuvre du Framework Hibernate

- Une classe de type JavaBean qui encapsule les données d'une table donnée nommée « classe de persistance » et contenant des annotations.
- Des propriétés de configuration, notamment des informations concernant la connexion à la base de données (url, login et pwd) contenant les mappings des classes.



Fichiers de configuration

Le fichier **hibernate.properties ou Hibernate.cfg.xml** sert à configurer l'accès à la base de données.

Les infos indispensables sont les suivantes :

- **hibernate.connection.driver_class** = Le chemin vers le driver JDBC,
- **hibernate.connection.url** = Le chemin d'accès à la base,
- **hibernate.connection.username** = Le nom d'utilisateur de la connexion,
- **hibernate.connection.password** = Le mot de passe de la connexion,

Fichiers de configuration

```
<hibernate-configuration>
 <session-factory>
  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ecole</property>
  <property name="hibernate.connection.username">root</property>
  <property name="hibernate.connection.password">root</property>
   <mapping class="classes.metiers.Produit"/>
 </session-factory>
</hibernate-configuration>
```

Les Classes de données

Une classe de données est un Javabean qui va encapsuler les propriétés de la table dans ses champs private mappés avec @ et contient des getters et setters.

@Entity
cette classe Java correspond à une table relationnelle.
@Entity(name = "ProduitEntity")
Si name non définit, par défaut c'est le nom de la classe

```
@Table(
   name = "PRODUITS",
   schema = "GESTION", //schéma de base de données
   catalog = "STOCKS", //database name
   uniqueConstraints = {
      @UniqueConstraint(name = "UK_PRODUIT_REF",
columnNames = {"REF"})
   },
)
```

```java
@Entity
@Table(name = "produit")
public class Produit {
    @Id
    @Basic(optional = false)
    @Column(name = "reference")
    private String ref;
    @Column(name = "nom_produit")
    private String nom;
    @Column(name = "prix")
    private double prix;
    @Column
    private int stock;
```

```
@Id
@Basic(Optional=false)
@GeneratedValue(strategy =
GenerationType.IDENTITY ) («auto-increment »)
@Column(
   name = "NOM_COLONNE",
   nullable = false,
   unique = true,
   length = 100,
   precision = 10,
   scale = 2,
   insertable = true,
   updatable = true,
   columnDefinition = "DECIMAL(10,2)",
   table = "AUTRE_TABLE"
)
```

Manipuler un produit dans la DB

Création d'un objet **Configuration Hibernate**.
•new Configuration() → objet qui représente la config Hibernate.
•.configure() → charge le fichier **hibernate.cfg.xml** par défaut (situé dans src/main/resources).

SessionFactory à partir de la config → c'est une usine qui produit des sessions de connexion à la base de données

La **Session Hibernate** correspond à une connexion avec la base de données.
Elle permet d'exécuter des transactions et d'interagir avec les entités (insert, update, delete, requêtes…).

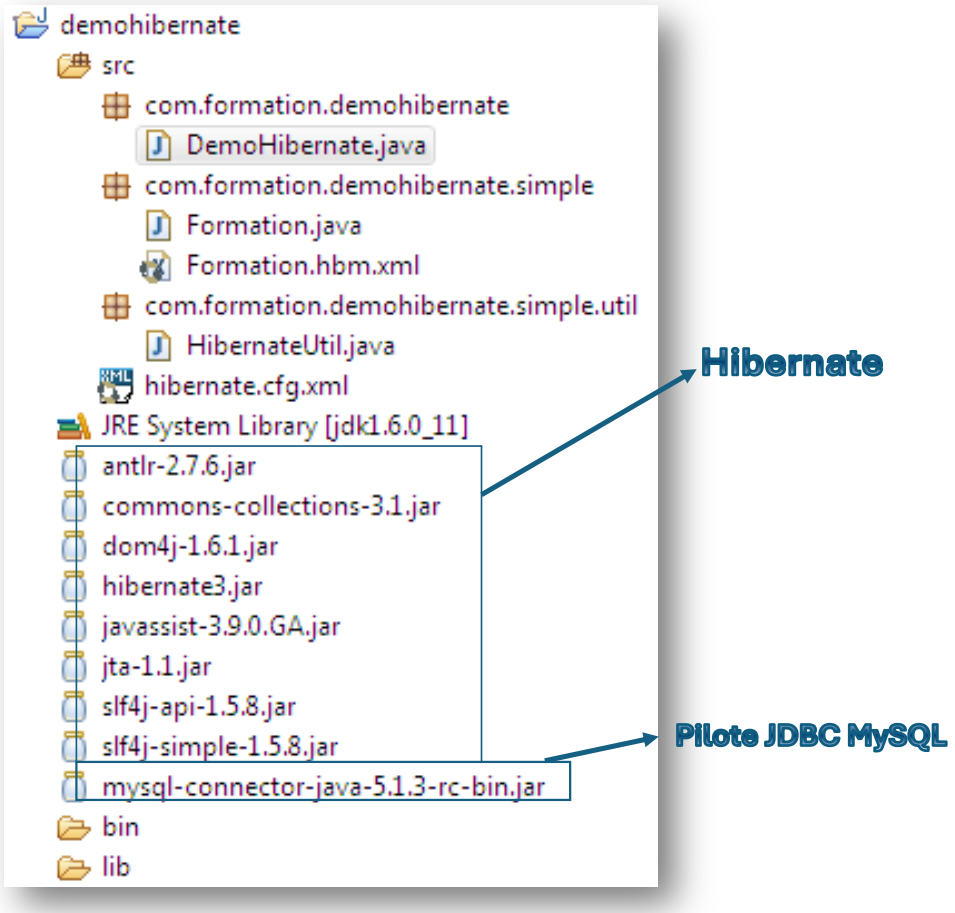
•openSession() → Ouvre une nouvelle session.
•close()→ Ferme la session et libère la connexion JDBC.

Session.persist(p) → insert
Session.save(p) → insert (return id)
Session.update(p)→ mettre à jour
Session.merge(p) → insert or update
Session.delete(p) → delete
Session.remove(p) → delete
Session.get(class,id)→ charge un objet de type class à partir de la base de données. On peut aussi utiliser laod()

```
Configuration cfg = new Configuration().configure(); // lit hibernate.c
SessionFactory sf = cfg.buildSessionFactory();
Session session = sf.openSession();
try {
    session.beginTransaction();
    session.persist(this);
    session.getTransaction().commit();
} catch (Exception ex) {
    System.out.println("Erreur d'enregisterment du produit");
    ex.printStackTrace();
}finally{
    session.close();
    sf.close();
}
```

Première Application

- Préparer le projet:
  - Au premier lieu, nous avons besoin des librairies Hibernate et de copier des fichiers .jar dans le répertoire lib de votre projet,
  - Pour travailler avec une base de données MySQL, nous aurons besoin également du connecteur JDBC de MySQL



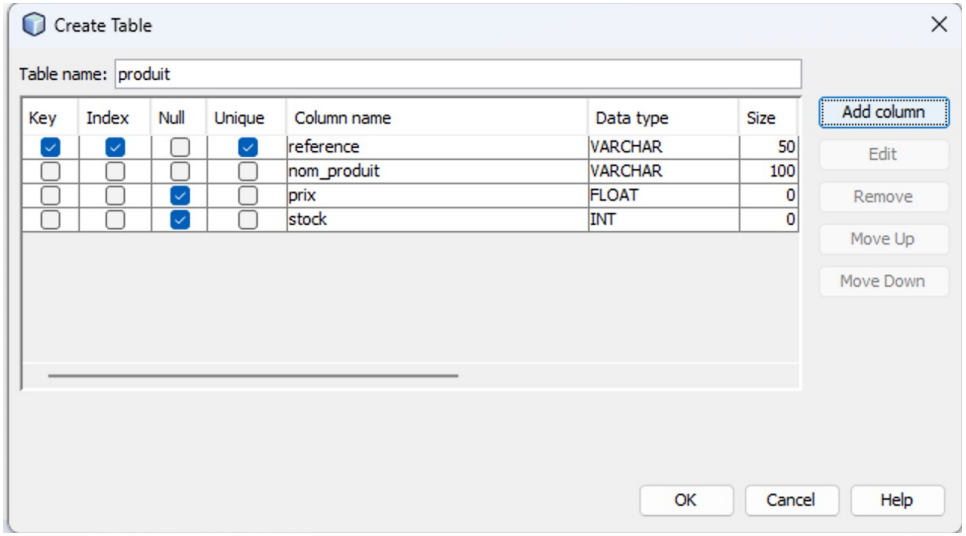
Exemple d’application

Création de la base de données « magasin »

```
create database magasin;
```

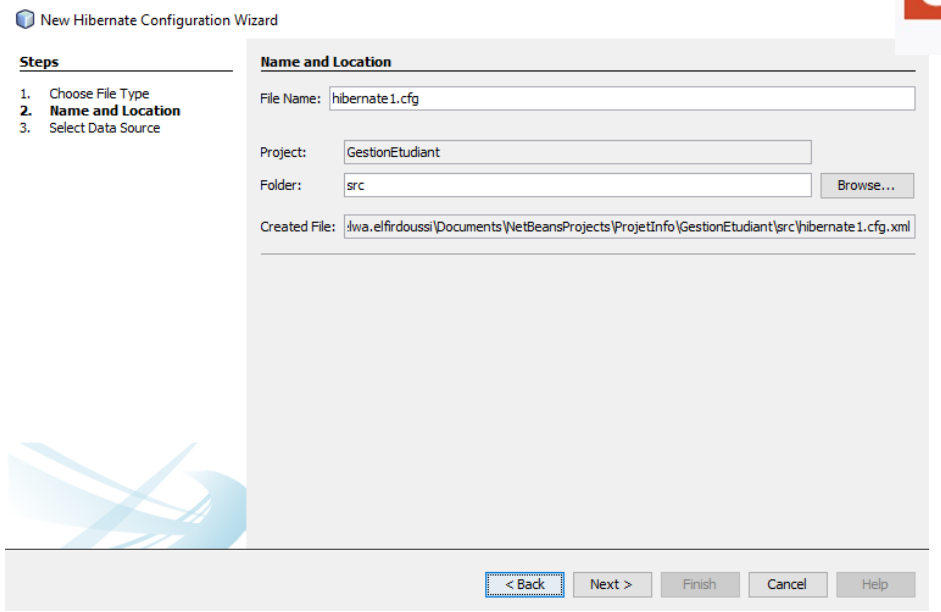
Création de la table « produit »

Attention, la table doit avoir une clé primaire



On va commencer par créer la table dans la base de données

Configurer Hibernate : Hibernate.cfg.xml




```
<hibernate-configuration>
 <session-factory>
  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/magasin</property>
  <property name="hibernate.connection.username">root</property>
  <property name="hibernate.connection.password">root</property>
 </session-factory>
</hibernate-configuration>
```

Classe Produit

Attention :

- annotation
- Getter & Setter générés

```java
@Entity
@Table(name = "produit")
public class Produit {
    @Id
    @Basic(optional = false)
    @Column(name = "reference")
    private String ref;
    @Column(name = "nom_produit")
    private String nom;
    @Column(name = "prix")
    private double prix;
    @Column
    private int stock;
```

```java
public String getRef() {
    return ref;
}

public void setRef(String ref) {
    this.ref = ref;
}

public String getNom() {
    return nom;
}

public void setNom(String nom) {
    this.nom = nom;
}

public double getPrix() {
    return prix;
}

public void setPrix(double prix) {
    this.prix = prix;
}

public int getStock() {
    return stock;
}

public void setStock(int stock) {
    this.stock = stock;
}
```

Application Test

- Il est temps de charger et de stocker quelques objets produits, mais d'abord nous devons compléter la configuration avec du code d'infrastructure.
- Nous devons démarrer Hibernate. Ce démarrage inclut le chargement de la configuration et la construction d'un objet SessionFactory global et le stocker quelque part facile d'accès dans le code de l'application.
- Une SessionFactory peut ouvrir des nouvelles Sessions.
- Une Session représente une unité de travail la SessionFactory est un objet global instancié une seule fois.

Créer la couche service

```
 */
public class ObjetService {


    public static Session openSession(){
        Configuration confHibernate = new Configuration();
        confHibernate.configure();
        SessionFactory sf = confHibernate.buildSessionFactory();
        Session session = sf.openSession();
        return session;
    }

    public static void closeSession(Session session){
        session.close();
    }

    public static void insert(Session session, Object obj) {
        try {
            Transaction tx = session.beginTransaction();
            session.persist(obj);
            tx.commit();
        } catch (Exception ex) {
            System.out.println("Erreur d'enregisterment du produit");
            ex.printStackTrace();
        }
    }
```

```
    public static void update(Session session, Object obj) {
        try {
            Transaction tx = session.beginTransaction();
            session.update(obj);
            tx.commit();
        } catch (Exception ex) {
            System.out.println("Erreur d'enregisterment du produit");
            ex.printStackTrace();
        }
    }

    public static void delete(Session session, Object obj) {
        try {
            Transaction tx = session.beginTransaction();
            session.update(obj);
            tx.commit();
        } catch (Exception ex) {
            System.out.println("Erreur d'enregisterment de l'objet");
            ex.printStackTrace();
        }
    }

    public static  void merge(Session session, Object obj) {
        try {
            Transaction tx = session.beginTransaction();
            session.merge(obj);
            tx.commit();
        } catch (Exception ex) {
            System.out.println("Erreur d'enregisterment du produit");
            ex.printStackTrace();
        }
    }
```

Créer des produits dans la méthode main

```java
public static void main(String[] args) {

    String addProduit = "yes";
    int i = 0;

    Session session = ObjetService.openSession();

    while(addProduit.equals("yes")) {
        Produit p = new Produit();
        Scanner scp = new Scanner(System.in);
        System.out.println(" Veuillez saisir le nom du produit N° " + i+1);
        p.setNom(scp.nextLine());

        System.out.println(" Veuillez saisir la référence du produit N° " + i+1);
        p.setRef(scp.nextLine());

        System.out.println(" Veuillez saisir le prix du produit N° " + i+1);
        p.setPrix(scp.nextDouble());

        System.out.println(" Veuillez saisir le stock du produit N° " + i+1);
        p.setStock(scp.nextInt());

        ObjetService.merge(session, p);

        Scanner sc = new Scanner(System.in);
        System.out.println("voulez vous continuer à saisir des produits taper yes ou no");
        addProduit = sc.nextLine();
        i++;

    }
```

Après Exécution

select * from produit ×

Page Size: 20 | Total Rows: 7 Page: 1 of 1 | Matching Rows:

#	reference	nom_produit	prix	stock
1	12Z	12Z	23.0	2
2	4333	23	6.0	7
3	AZ23	AZ23	33322.0	23
4	R45	R45	24.0	7
5	RT43	RT43	678.0	87
6	TTRR	Z45	60.0	78
7	Z23	Z23	23.0	23

Les autres opérations

```java
public static void update(Session session, Object obj) {
    try {
        Transaction tx = session.beginTransaction();
        session.update(obj);
        tx.commit();
    } catch (Exception ex) {
        System.out.println("Erreur d'enregisterment du produit");
        ex.printStackTrace();
    }
}

public static void delete(Session session, Object obj) {
    try {
        Transaction tx = session.beginTransaction();
        session.update(obj);
        tx.commit();
    } catch (Exception ex) {
        System.out.println("Erreur d'enregisterment de l'objet");
        ex.printStackTrace();
    }
}

public static  void merge(Session session, Object obj) {
    try {
        Transaction tx = session.beginTransaction();
        session.merge(obj);
        tx.commit();
    } catch (Exception ex) {
        System.out.println("Erreur d'enregisterment du produit");
        ex.printStackTrace();
    }
}
```

Exercice

Dans la méthode main créer un menu et programmer les instructions liées a chaque entrée du menu :

Implémenter les CRUD (Create, Read, Update et Delete)

1: Créer un produit
2: Afficher un produit à partir de la référence
3: Modifier le stock d'un produit
4: Supprimer un produit
8: Quitter (System.exit())

HQL : Hibernate Query Language

```java
public static List<Produit> listproduit(Session session) {
    List<Produit> produits = session.createQuery("FROM Produit").list();
    return produits;
}
```

CreateQuery : permet d'exécuter des requêtes **HQL**, comme du SQL, elle utilise les noms des classes et des attributs.

```
List<Produit> produits =
session.createQuery("FROM Produit p WHERE p.prix > 100").list();
session.createQuery("FROM Produit p ORDER BY p.prix ASC ").list();
session.createQuery("SELECT p.nom, SUM(p.stock) FROM Produit p GROUP BY p.nom").list();
session.createQuery("FROM Produit p WHERE p.nom = :nom").setParameter("nom",  "DELL").list();
session.createQuery("FROM Produit p WHERE p.stock < :stock").setParameter(  "stock",  10).list();
```

- createQuery("FROM Produit") utilise le **nom de la classe**, pas la table SQL. Et renvoie une liste des objets de type Produit
- Paramètres → :nom, :prix : on utilise setParameter pour donner les valeurs
- ORDER BY, GROUP BY, HAVING → fonctionnent comme en SQL.

- Le SELECT ne renvoie pas des entités, on obtient des tableaux d'objets (Object[]).

Exercice

Dans votre méthode main ajouter un menu 6 pour lister les produits et un menu et programmer les instructions liées a chaque entrée du menu :

1: Créer un produit
2: Afficher un produit à partir de la référence
3: Modifier le stock un produit
4: Supprimer un produit
5: Lister les produits par ordre de prix
6: Lister les produits ayant un stock inférieur à un stock saisi en paramètre
7: Lister le nom et le prix des produits ayant un stock supérieur à un stock maximum
8: Quitter (System.exit())

Post Lab & Pré Lab

Postlab (POO en Java)

- ❑ Comment Hibernate gère les clés étrangères.
- ❑ EX : cas de la table mouvementDeStock
- ❑ Que veut dire les annotations suivantes :

  a) @OneToMany

  b) @ManyToOne

  c) @JoinColumn

Prélab (Gestion des interfaces Graphiques JAVA avec Java FX)

- ❑ XML : définition, structure.
- ❑ C'est quoi une interface Graphique
- ❑ C'est les différents API permettant de créer des interfaces graphiques en JAVA
- ❑ Quelle est la structure d'un projet JAVAFX
- ❑ Que contient un fichier FXML
- ❑ Comment lier un fichier FXML avec du code JAVA