# Formation PHP Orienté Objet

## Projet : Système de gestion de vente en ligne

**Prérequis**

- Connaissances de base en PHP
- Compréhension des concepts de base de la POO
- Familiarité avec les bases de données SQL

**Objectifs pédagogiques**

- ➔ Maîtriser les concepts fondamentaux de la POO en PHP
- ➔ Comprendre et implémenter les design patterns
- ➔ Acquérir les bonnes pratiques de développement professionnel

**Structure du projet**

Jour 1 : Fondations
Jour 2 : Relations et hiérarchies
Jour 3 : Organisation et patterns
Jour 4 : Persistance des données
Jour 5 : Concepts avancés (optionnel)

**Livrables attendus**

Le seul livrable attendu est le code source complet, qui devra être transmis selon l'une des deux options suivantes **avant le dimanche 1er décembre 2023 minuit** :

Archive ZIP contenant l'intégralité du projet à envoyer sur Discord
**OU**
URL du repository (GitHub, GitLab, etc.) à envoyer sur Discord

**Environnement technique**

- ☐ PHP 8.x
- ☐ MySQL/MariaDB
- ☐ Composer
- ☐ IDE (VSCode recommandé)
- ☐ Git (optionnel)

**JOUR 1 - Après-midi : Fondations**

**Contexte**

En travaillant sur un projet de système de gestion de vente en ligne, vous allez apprendre à concevoir et structurer des applications complexes en utilisant des bonnes pratiques de développement logiciel.

**1. Configuration de l'environnement**

- Installation de WAMP (version supportant PHP 8.x)
- Visual Studio Code avec les extensions recommandées :
    - PHP Intelephense
    - PHP DocBlocker
    - PHP Namespace Resolver
- Création de la structure de base du projet

**2. Création des premières classes dans le dossier src/Entity/**

| **Produit** |
| --- |
| ❖ id (entier, nullable)<br>❖ nom (chaîne)<br>❖ description (chaîne)<br>❖ prix (nombre décimal)<br>❖ stock (entier) |
| ➢ Un constructeur avec les paramètres nécessaires<br>➢ Getters et setters pour chaque propriété<br>➢ calculerPrixTTC() : float : Retourne le prix TTC en appliquant une TVA de 20 %<br>➢ verifierStock(int $quantite) : bool : Vérifie si le stock est suffisant pour la quantité demandée. Retourne true si le stock est suffisant, sinon false |

| **Utilisateur** |
| --- |
| ❖ id (entier, nullable)<br>❖ nom (chaîne)<br>❖ email (chaîne)<br>❖ motDePasse (chaîne)<br>❖ dateInscription (DateTime) |
| ➢ Un constructeur avec les paramètres nécessaires<br>➢ Getters et setters pour chaque propriété<br>➢ verifierMotDePasse(string $motDePasse) : bool : Vérifie si le mot de passe fourni correspond à celui de l'utilisateur<br>➢ mettreAJourProfil(string nom, string email, string $motDePasse) : void : Met à jour les informations de l'utilisateur avec validation appropriée |

### 3. Application de l'encapsulation

- Toutes les propriétés doivent être privées
- Accès uniquement via getters/setters
- Validation des données dans les setters

Validation des données

Implémentez ces règles de validation :

- ➔ Prix : doit être positif
- ➔ Stock : doit être positif ou nul
- ➔ Email : doit être une adresse email valide
- ➔ Mot de passe : minimum 8 caractères
- ➔ Nom : ne doit pas être vide

Documentation

Chaque classe et méthode doit être documentée avec PHPDoc :

- Description de la classe
- Description des méthodes
- Types des paramètres et valeurs de retour
- Exceptions potentielles

**Tests à réaliser**

Créez un fichier **public/index.php** pour tester :

- Création d'instances de produits et d'utilisateurs
- Modification des propriétés via setters
- Calcul de prix TTC
- Vérification de stock
- Validation du mot de passe
- Mise à jour de profil

**Structure du projet à la fin du jour 1**

```
projet-vente-en-ligne/
│
├── src/
│   ├── Entity/
│   │   ├── Produit.php
│   │   └── Utilisateur.php
│
└── public/
    └── index.php
```

**JOUR 2 - Après-midi : Relations et hiérarchies**

### 4. Hiérarchie des produits

➔ Classe abstraite Produit

Transformez votre classe Produit du jour 1 en classe abstraite avec :

- Toutes les propriétés et méthodes communes aux produits
- Une méthode abstraite calculerFraisLivraison()
- Une méthode abstraite afficherDetails()

➔ Classes dérivées à implémenter

| **ProduitPhysique** |
| --- |
| ❖ poids (float, kg)<br>❖ longueur (float, cm)<br>❖ largeur (float, cm)<br>❖ hauteur (float, cm) |
| ➢ calculerVolume() : float : Calcule le volume du produit en cm³ (longueur × largeur × hauteur).<br>➢ calculerFraisLivraison() : float : Calcule les frais de livraison basés sur le poids du produit. |

| **ProduitNumerique** |
| --- |
| ❖ lienTelechargement (string)<br>❖ tailleFichier (float, MB)<br>❖ formatFichier (string) |
| ➢ genererLienTelechargement() : string : Génère et retourne un lien de téléchargement unique pour le produit.<br>➢ calculerFraisLivraison() : float : Retourne toujours 0, car il n'y a pas de frais de livraison pour les produits numériques. |

| **ProduitPerissable** |
| --- |
| ❖ dateExpiration (DateTime)<br>❖ temperatureStockage (float) |
| ➢ estPerime() : bool : Vérifie si le produit est périmé par rapport à la date actuelle.<br>➢ calculerFraisLivraison() : float : Retourne les frais de livraison avec une majoration de 5 $ pour les produits frais. |

## 5. Relations entre classes

| **Categorie** |
|---|
| ❖ id (int)<br>❖ nom (string)<br>❖ description (string)<br>❖ produits (array de Produit) |
| ➢ ajouterProduit(Produit $produit) : void : Ajoute un produit à la catégorie<br>➢ retirerProduit(Produit $produit) : void : Retire un produit de la catégorie<br>➢ listerProduits() : array : Retourne un tableau des produits dans la catégorie |

| **Panier** |
|---|
| ❖ articles : array : Tableau associatif où la clé est l'ID du produit et la valeur est un tableau contenant le produit et sa quantité.<br>❖ dateCreation : DateTime |
| ➢ ajouterArticle(Produit produit, intquantite) : void : Ajoute un produit au panier avec une quantité spécifiée. Gère l'augmentation de la quantité si le produit est déjà présent.<br>➢ retirerArticle(Produit produit, intquantite) : void : Retire une quantité spécifiée d'un produit du panier. Retire complètement le produit si la quantité tombe à zéro ou en dessous.<br>➢ vider() : void : Vide le panier de tous ses articles.<br>➢ calculerTotal() : float : Calcule et retourne le total du panier en tenant compte du prix TTC de chaque produit.<br>➢ compterArticles() : int : Retourne le nombre total d'articles dans le panier. |

## 6. Hiérarchie des utilisateurs

➔ Modification de la Classe Utilisateur

Transformez la classe Utilisateur en classe abstraite avec :

- Une méthode abstraite afficherRoles()
- Une propriété protégée roles (array)

➔ Classes Dérivées à Implémenter

| **Client** |
|---|
| ❖ adresseLivraison (string)<br>❖ panier (Panier) |
| ➢ passerCommande() : void : Crée une commande à partir des articles présents dans le panier. (vide pour l'instant)<br>➢ consulterHistorique() : array : Retourne l'historique des commandes passées par le client. (vide pour l'instant) |

| **Admin** |
| --- |
| ❖ niveauAcces (int)<br>❖ dernieresConnexion (DateTime) |
| ➢ gererUtilisateurs() : void : Permet de gérer les utilisateurs du système (ajout, modification, suppression). (vide pour l'instant)<br>➢ accederJournalSysteme() : array : Accéder aux logs du système pour les analyses d'audit. (vide pour l'instant) |

| **Vendeur** |
| --- |
| boutique (string)<br>commission (float) |
| ➢ ajouterProduit(Produit $produit) : void : Ajoute un produit à la boutique du vendeur. (vide pour l'instant)<br>➢ gererStock(Produit produit, intquantite) : void : Gère le stock de produits dans la boutique du vendeur. (vide pour l'instant) |

**Tests à réaliser**

Dans votre fichier de test, créez des scénarios pour :

- Création de différents types de produits
- Manipulation du panier
- Création et gestion des différents types d'utilisateurs

**Structure du projet à la fin du jour 2**

```
projet-vente-en-ligne/
│
├── src/
│   ├── Entity/
│   │   ├── Produit/
│   │   │   ├── Produit.php  # Classe abstraite
│   │   │   ├── ProduitPhysique.php
│   │   │   ├── ProduitNumerique.php
│   │   │   └── ProduitPerissable.php
│   │   ├── Utilisateur/
│   │   │   ├── Utilisateur.php  # Classe abstraite
│   │   │   ├── Client.php
│   │   │   ├── Admin.php
│   │   │   └── Vendeur.php
│   │   ├── Categorie.php
│   │   └── Panier.php
│
└── public/
    └── index.php
```

**JOUR 3 - Après-midi : Organisation et patterns**

### 7. Autoloading

- Installer Composer
- Configurer composer.json
- Utilisation de PSR-4 : Assurez-vous que tous les namespaces et chemins sont correctement configurés dans composer.json
- Exécuter composer dump-autoload
- Inclure require 'vendor/autoload.php' dans index.php

### 8. Design Pattern : Factory

Créer la classe ProduitFactory dans App\Factory :

- Méthode principale creerProduit(string type, array data) : Crée un produit en fonction du type spécifié ("physique", "numerique", "perissable") et des données fournies. Valide le type et les données, retourne le produit créé ou lance une exception en cas de données invalides.
- Méthodes privées de création :
    - Valident les données spécifiques à chaque type
    - Créent et retournent l'instance appropriée
    - Lancent des exceptions si données invalides

### 9. Design Pattern : Singleton

Créer la classe ConfigurationManager dans App\Config :

- Gestion de l'instance unique :
    - Méthode getInstance() retournant l'instance unique
    - Constructeur privé
- Gestion de la configuration :
    - Charger la configuration (depuis un fichier ou tableau)
    - Méthodes get/set pour accéder aux paramètres
    - Validation des paramètres
- Paramètres à gérer :
    - TVA
    - Devise
    - Frais de livraison de base
    - Email de contact

*Pensez à mettre à jour vos anciennes méthodes afin d'utiliser le ConfigurationManager (exemple : la fonction Produit:calculerPrixTTC() avec la TVA).*

**Tests à réaliser**

Dans votre fichier de test, créez des scénarios pour :

- Tester le bon fonctionnement de l'autoloading
- Tester la factory de produits
- Tester le bon fonctionnement du ConfigurationManager

**Structure du projet à la fin du jour 3**

```
projet-vente-en-ligne/
│
├── src/
│   ├── Factory/
│   │   └── ProduitFactory.php
│   ├── Config/
│   │   └── ConfigurationManager.php
│   ├── Entity/
│   │   ├── Produit/
│   │   │   ├── Produit.php
│   │   │   ├── ProduitPhysique.php
│   │   │   ├── ProduitNumerique.php
│   │   │   └── ProduitPerissable.php
│   │   ├── Utilisateur/
│   │   │   ├── Utilisateur.php
│   │   │   ├── Client.php
│   │   │   ├── Admin.php
│   │   │   └── Vendeur.php
│   │   ├── Categorie.php
│   │   └── Panier.php
│
├── vendor/  # Généré par Composer
│
├── composer.json
│
└── public/
    └── index.php
```

**JOUR 4 - Après-midi : Persistance des données**

## 10. Intégration de PDO

Création de la classe DatabaseConnection (Singleton) :

- Créez une classe DatabaseConnection dans le dossier App\Database.
- Appliquez le pattern Singleton pour garantir une seule instance de connexion PDO dans l'application.
- Définissez les paramètres de connexion nécessaires (hôte, base de données, utilisateur, mot de passe).
- Intégrez la gestion de ces paramètres via un fichier de configuration.
- Assurez-vous de lever des exceptions en cas de problème de connexion.

## 11. CRUD

Création des classes Repository :

- ProduitRepository
- UtilisateurRepository
- CategorieRepository

Implémentez les méthodes suivantes pour interagir avec les tables en utilisant des entités :

- create(Produit $produit) : int
  Ajoute un nouvel enregistrement à partir d'une entité Produit et retourne l'ID du nouvel enregistrement.

- read(int $id) : ?Produit
  Récupère un enregistrement par son ID et retourne l'entité Produit correspondante, ou null si aucun produit n'est trouvé.

- update(Produit $produit) : void
  Modifie un enregistrement existant en utilisant les données de l'entité Produit fournie.

- delete(int $id) : void
  Supprime un enregistrement par son ID.

- findAll() : array
  Récupère tous les enregistrements sous forme de tableau d'entités Produit.

- findBy(array $criteria) : array
  Recherche des enregistrements selon des critères spécifiques et retourne les résultats sous forme de tableau d'entités Produit.

**Tests à réaliser**

Dans votre fichier de test, créez des scénarios pour :

- Tester si la connexion à la base de données est établie sans erreur.
- Tester la création, la lecture, la mise à jour et la suppression des enregistrements pour chaque repository.
- Vérifier que les données sont correctement manipulées dans la base de données.

**Structure du projet à la fin du jour 4**

```
projet-vente-en-ligne/
│
├── src/
│   ├── Database/
│   │   └── DatabaseConnection.php
│   ├── Repository/
│   │   ├── ProduitRepository.php
│   │   ├── UtilisateurRepository.php
│   │   └── CategorieRepository.php
│   ├── Factory/
│   │   └── ProduitFactory.php
│   ├── Config/
│   │   └── ConfigurationManager.php
│   ├── Entity/
│   │   ├── Produit/
│   │   │   ├── Produit.php
│   │   │   ├── ProduitPhysique.php
│   │   │   ├── ProduitNumerique.php
│   │   │   └── ProduitPerissable.php
│   │   ├── Utilisateur/
│   │   │   ├── Utilisateur.php
│   │   │   ├── Client.php
│   │   │   ├── Admin.php
│   │   │   └── Vendeur.php
│   │   ├── Categorie.php
│   │   └── Panier.php
│
├── vendor/  # Généré par Composer
│
├── composer.json
│
└── public/
    └── index.php
```

**JOUR 5 - Après-midi : Concepts avancés (Optionnel)**

### 12. Architecture MVC

Mise en œuvre de l'architecture MVC :

- Création des contrôleurs :
  - Créez des classes contrôleurs pour gérer les requêtes HTTP et orchestrer les interactions entre le modèle et la vue.
  - Exemples de contrôleurs : ProduitController, UtilisateurController, etc.
  - Chaque contrôleur doit contenir des méthodes pour gérer les actions (e.g., afficher la liste des produits, gérer les utilisateurs).

- Création des vues :
  - Développez des fichiers de vue pour présenter les données à l'utilisateur.
  - Utilisez des templates HTML/PHP pour générer l'interface utilisateur.
  - Assurez-vous que les vues sont séparées des contrôleurs pour respecter la séparation des responsabilités.

- Routing simple :
  - Implémentez un système de routage simple pour diriger les requêtes vers les contrôleurs appropriés.
  - Définissez des routes pour les différentes actions de l'application (e.g., /produits, /utilisateurs).

### 13. Patterns avancés

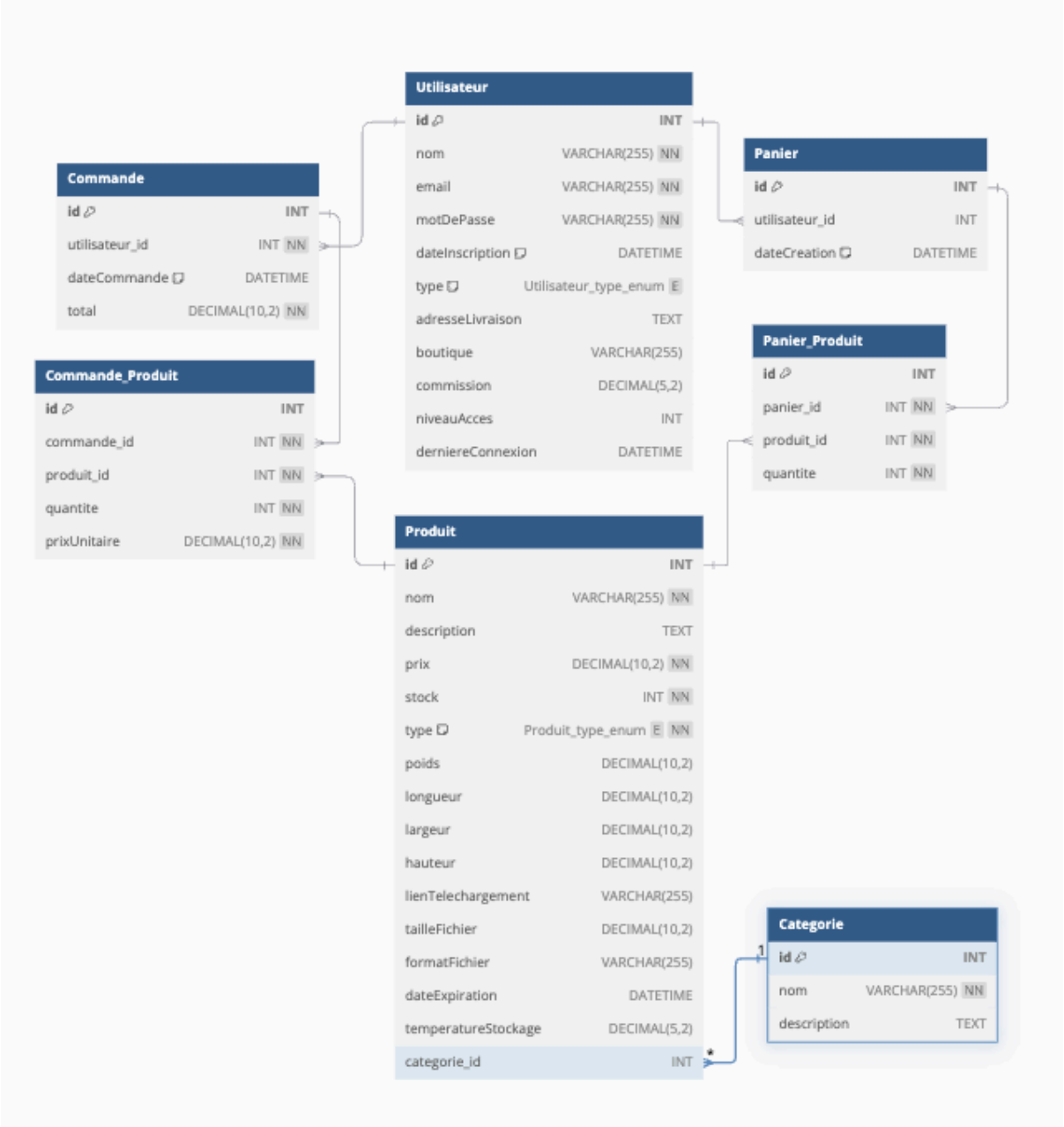
Implémentation **au choix** parmi les patterns suivants :

- Décorateur pour les produits : Implémenter le pattern Décorateur pour ajouter dynamiquement des fonctionnalités ou attributs aux produits sans modifier leurs classes.
- Façade pour simplifier l'API : Créez une façade pour simplifier l'utilisation de l'application en offrant une interface plus simple pour des sous-systèmes complexes.
- Proxy pour la mise en cache : Utilisez le pattern Proxy pour ajouter une couche de mise en cache autour des objets fréquemment demandés, comme les produits.
- Itérateur pour les collections de produits : Implémentez un itérateur pour parcourir facilement les collections de produits, facilitant la manipulation des listes de produits.
- State pour les états de commande : Appliquez le pattern State pour gérer les différents états d'une commande (e.g., en cours, expédiée, livrée).
- Observer pour les notifications : Utilisez le pattern Observer pour notifier automatiquement les utilisateurs ou administrateurs des changements importants (e.g., nouvelle commande, changement de stock).

**Structure du projet (MVC) à la fin du jour 5**

```
projet-vente-en-ligne/
│
├── src/
│   ├── Controller/
│   │   ├── ProduitController.php
│   │   └── UtilisateurController.php
│   ├── Database/
│   │   └── DatabaseConnection.php
│   ├── Repository/
│   │   ├── ProduitRepository.php
│   │   ├── UtilisateurRepository.php
│   │   └── CategorieRepository.php
│   ├── Factory/
│   │   └── ProduitFactory.php
│   ├── Config/
│   │   └── ConfigurationManager.php
│   ├── Entity/
│   │   ├── Produit/
│   │   │   ├── Produit.php
│   │   │   ├── ProduitPhysique.php
│   │   │   ├── ProduitNumerique.php
│   │   │   └── ProduitPerissable.php
│   │   ├── Utilisateur/
│   │   │   ├── Utilisateur.php
│   │   │   ├── Client.php
│   │   │   ├── Admin.php
│   │   │   └── Vendeur.php
│   │   ├── Categorie.php
│   │   └── Panier.php
│
├── views/
│   ├── produit/
│   │   └── list.php
│   └── utilisateur/
│       └── profile.php
│
├── vendor/  # Généré par Composer
│
├── composer.json
│
└── public/
    ├── index.php
    └── .htaccess
```

## Annexe 1 : Diagramme de classes



### Annexe 2 : Script SQL de création des tables

```
CREATE TABLE Utilisateur (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    motDePasse VARCHAR(255) NOT NULL,
    dateInscription DATETIME DEFAULT CURRENT_TIMESTAMP,
    type ENUM('client', 'admin', 'vendeur') DEFAULT 'client',
    adresseLivraison TEXT NULL,
    boutique VARCHAR(255) NULL,
    commission DECIMAL(5,2) NULL,
    niveauAcces INT NULL,
    derniereConnexion DATETIME NULL
);

CREATE TABLE Categorie (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(255) NOT NULL,
    description TEXT
);

CREATE TABLE Produit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(255) NOT NULL,
    description TEXT,
    prix DECIMAL(10, 2) NOT NULL,
    stock INT NOT NULL,
    type ENUM('physique', 'numerique', 'perissable') NOT NULL,
    poids DECIMAL(10, 2) NULL,
    longueur DECIMAL(10, 2) NULL,
    largeur DECIMAL(10, 2) NULL,
    hauteur DECIMAL(10, 2) NULL,
    lienTelechargement VARCHAR(255) NULL,
    tailleFichier DECIMAL(10, 2) NULL,
    formatFichier VARCHAR(255) NULL,
    dateExpiration DATETIME NULL,
    temperatureStockage DECIMAL(5, 2) NULL,
    categorie_id INT,
    FOREIGN KEY (categorie_id) REFERENCES Categorie(id)
);

CREATE TABLE Panier (
    id INT AUTO_INCREMENT PRIMARY KEY,
    utilisateur_id INT,
    dateCreation DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (utilisateur_id) REFERENCES Utilisateur(id)
);
```

```
CREATE TABLE Panier_Produit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    panier_id INT NOT NULL,
    produit_id INT NOT NULL,
    quantite INT NOT NULL,
    FOREIGN KEY (panier_id) REFERENCES Panier(id),
    FOREIGN KEY (produit_id) REFERENCES Produit(id)
);

CREATE TABLE Commande (
    id INT AUTO_INCREMENT PRIMARY KEY,
    utilisateur_id INT NOT NULL,
    dateCommande DATETIME DEFAULT CURRENT_TIMESTAMP,
    total DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (utilisateur_id) REFERENCES Utilisateur(id)
);

CREATE TABLE Commande_Produit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    commande_id INT NOT NULL,
    produit_id INT NOT NULL,
    quantite INT NOT NULL,
    prixUnitaire DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (commande_id) REFERENCES Commande(id),
    FOREIGN KEY (produit_id) REFERENCES Produit(id)
);

-- Optionally, create tables for logs, sessions, or additional
features as needed.
```