TP fil rouge : Application JDBC de gestion des utilisateurs

Dr. LO

10 novembre 2025

Objectifs du TP

L'objectif de ce TP est de développer, pas à pas, une petite application Java en ligne de commande permettant de gérer des **utilisateurs** stockés dans une base de données relationnelle.

À l'issue du TP, l'étudiant devra être capable de :

— Configurer une connexion JDBC (sans Maven) et tester la connexion.

— Utiliser `PreparedStatement` et `ResultSet` pour exécuter des requêtes paramétrées.

— Récupérer les clés générées (*generated keys*) après un `INSERT`.

— Organiser le code d'accès aux données sous forme de **DAO** (Data Access Object).

— Mettre en place un petit menu console pour réaliser un CRUD complet.

— Manipuler des transactions (`commit`/`rollback`).

1 Étape 1 : Mise en place de la base de données

Travail à faire

1) Créez une base de données, par exemple nommée `coursjdbc`.
2) Créez la table `utilisateurs` avec la structure suivante :

```
CREATE TABLE IF NOT EXISTS utilisateurs (
  id    BIGINT PRIMARY KEY AUTO_INCREMENT,
  nom   VARCHAR(100) NOT NULL,
  age   INT CHECK (age >= 0),
  email VARCHAR(150) UNIQUE
);
```

3) Insérez quelques lignes de test :

```
INSERT INTO utilisateurs (nom, age, email) VALUES
('Alice', 22, 'alice@example.com'),
('Bob',   30, 'bob@example.com');
```

2 Étape 2 : Création du projet Java (sans Maven) et ajout du driver JDBC

Travail à faire

1) Créez un projet **Java Application** classique dans votre IDE (NetBeans, IntelliJ, Eclipse, etc.).
2) Téléchargez le driver JDBC correspondant à votre SGBD (par exemple `mysql-connector-j-....jar`) et ajoutez-le au *classpath* du projet (Libraries → Add JAR).
3) Créez une classe `TestConnexion` qui :

— construit l'URL JDBC, le nom d'utilisateur et le mot de passe ;
— ouvre une connexion à l'aide de `DriverManager.getConnection` dans un bloc `try-with-resources` ;
— affiche un message « Connexion réussie » en cas de succès ;
— affiche le nom et la version du SGBD à l'aide de `DatabaseMetaData`.

3 Étape 3 : Lecture avec `PreparedStatement` et `ResultSet`

Objectif

Réaliser une requête `SELECT` paramétrée pour lister les utilisateurs ayant un âge minimal donné.

Travail à faire

1) Créez une classe `ListerUtilisateurs`.
2) Dans la méthode `main`, écrivez le code qui :
   — demande à l'utilisateur un âge minimum (via `Scanner`) ;
   — prépare la requête suivante avec un `PreparedStatement` :

```
SELECT id, nom, age, email
FROM utilisateurs
WHERE age >= ?
ORDER BY age;
```

   — fixe le paramètre `?` avec la valeur saisie ;
   — exécute la requête et parcourt le `ResultSet` avec `while (rs.next())` ;
   — affiche chaque ligne sous la forme `id - nom (age) <email>`.
3) Expliquez en quoi `PreparedStatement` protège contre l'injection SQL par rapport à un simple `Statement` avec concaténation de chaînes.

4 Étape 4 : Insertion et clés générées (*generated keys*)

Objectif

Insérer un nouvel utilisateur et récupérer l'identifiant auto-généré par la base.

Travail à faire

1) Créez une classe `CreerUtilisateur`.
2) Le programme doit :
   — demander au clavier le `nom`, l'`age` et l'`email` d'un nouvel utilisateur ;
   — préparer la requête `INSERT` suivante avec `PreparedStatement` :

```
INSERT INTO utilisateurs(nom, age, email)
VALUES (?, ?, ?);
```

   — créer le `PreparedStatement` avec l'option

```
Statement.RETURN_GENERATED_KEYS
```

   — exécuter la requête avec `executeUpdate()` ;
   — récupérer l'identifiant généré à l'aide de `getGeneratedKeys()` et l'afficher.

5 Étape 5 : Mise en place d'un DAO `UtilisateurDao`

Objectif

Isoler le code d'accès à la base de données dans une couche DAO (Data Access Object).

Travail à faire

1) Créez une classe métier `Utilisateur` contenant les attributs `id`, `nom`, `age` et `email`, avec les accesseurs (`getters/setters`).
2) Créez une interface `UtilisateurDao` définissant au minimum les méthodes suivantes :

```
long create(Utilisateur u) throws SQLException;
Utilisateur findById(long id) throws SQLException;
List<Utilisateur> findAll() throws SQLException;
void update(Utilisateur u) throws SQLException;
void delete(long id) throws SQLException;
```

3) Créez une classe `UtilisateurDaoJdbc` qui implémente `UtilisateurDao` en utilisant l'API JDBC (`Connection`, `PreparedStatement`, `ResultSet`) et des blocs `try-with-resources`.

6 Étape 6 : Application console avec menu (CRUD complet)

Objectif

Utiliser le DAO dans une application principale qui propose un menu texte à l'utilisateur.

Travail à faire

1) Créez une classe `ApplicationConsole` avec une méthode `main`.
2) Mettez en place un menu similaire à :

```
1) Lister les utilisateurs
2) Créer un utilisateur
3) Rechercher un utilisateur par id
4) Mettre à jour un utilisateur
5) Supprimer un utilisateur
0) Quitter
```

3) En fonction du choix saisi, appelez les méthodes correspondantes du `UtilisateurDao` :
   — `findAll()` pour la liste,
   — `create(...)` pour l'ajout,
   — `findById(...)` puis `update(...)` pour la mise à jour,
   — `delete(...)` pour la suppression.
4) Veillez à ne plus écrire de SQL directement dans `main` : tout doit passer par le DAO.

7 Étape 7 : Transactions (optionnel mais recommandé)

Objectif

Manipuler `setAutoCommit(false)`, `commit()` et `rollback()` sur un exemple simple (virement entre comptes ou opération groupée).

Piste de travail

— Créez une table `compte` avec un identifiant, un nom et un solde.
— Écrivez un programme qui effectue un virement d'un compte A vers un compte B dans une transaction :
   — `UPDATE compte SET solde = solde - montant WHERE id = A;`
   — `UPDATE compte SET solde = solde + montant WHERE id = B;`

— Encadrez ces deux requêtes dans un bloc avec `setAutoCommit(false)`, puis validez avec `commit()` ou annulez avec `rollback()` en cas d'erreur.

Remarque : Vous pouvez enrichir le projet en ajoutant une interface graphique (Swing) ou une interface web (Servlet/JSP) qui réutilise le même DAO, afin de bien comprendre la séparation entre la couche présentation et la couche accès aux données.