Objectifs

L'utilitaire phpMyAdmin est certainement l'un des outils d'administration de base de données les plus connus. Il existe d'ailleurs un dérivé pour postgresql qui est installé ici pour vous. L'objectif de ce projet consiste à réaliser une mini-application web du même genre, avec les 4 méthodes du DML sur une table passée en paramètre.

Vous réaliserez ce projet avec Maven et rendrez sur Moodle une archive ZIP contenant l'ensemble du projet avec une dépendance sur tomcat de manière à pouvoir le lancer par les commandes

```
mvn clean package
mvn tomcat:run
```

1. Ecrire une servlet `Select.java` qui permet d'afficher n'importe quelle table dont le nom est passé en paramètre. La servlet doit bien sûr utiliser les `MetaData` pour connaitre le nombre et les noms de colonnes.
   Ex: `vide/servlet-Select?table=personne`
   On a besoin ici d'une seule servlet.

2. Ecrire une servlet `Insert.java` qui présente un formulaire permettant d'insérer une ligne dans une table dont le nom est passé en paramètre. Le formulaire n'est pas "en dur" mais calculé dynamiquement en fonction du nom de la table à l'aide de la Métabase. On utilisera avantageusement les méthodes `doGet` pour afficher le formulaire et `doPost` pour effectuer l'insertion
   Ex: `vide/servlet-Insert?table=personne`
   Deux méthodes ou deux servlets; l'une fabrique le formulaire HTML, l'autre exécute la requete. Utilisation de champs hidden

3. Mise en place d'une CSS
   Les Cascade Style Sheets (CSS) permettent de mettre de la forme dans les pages générées. Il est bien sûr possible de créer sa propre CSS, mais certaines sont très populaires. C'est le cas de Bootstrap initialement fourni par Twitter : `http://getbootstrap.com/`. Cette CSS est très bien documentée, avec de nombreux exemples. Comme pour toute CSS, il est possible soit d'installer les fichiers nécessaires dans sa propre archive, soit d'y faire référence. Nous prendrons cette seconde solution. Vos pages doivent donc contenir les lignes :

   ```
   <!Doctype html>
   <html lang="en">
     <head>
       <meta charset="utf-8">
       <meta http-equiv="X-UA-Compatible" content="IE=edge">
       <meta name="viewport" content="width=device-width, initial-scale=1">
       <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.m
     </head>
   ...
   ```

   Mettre en place un style minimal pour les tables, formulaires, boutons et requêtes utilisés dans ce TP (onglet `CSS` du site Bootstrap). Par exemple `class="alert alert-warning"` pour les requetes affichées, `class="table table-hover"` pour les tables ou `class="form-group row"` pour les formulaires

4. Modifiez les servlets `Select` et `Insert` de manière à ce que ces deux opérations puissent se faire dans une seule et même page. Le formulaire de saisie sera affiché en dernière ligne de la table Select, un bouton Insertion en dessous. L'insertion rappellera automatiquement la page `Select`
5. Créez une servlet `Delete.java` qui permet de détruire une ligne d'une table. La table sera passée en paramètre et la ligne sera repérée par l'ensemble de ses valeurs.(dans un premier temps, on pourra considérer pour simplifier que la première colonne correspond à la clé et qu'on ne passe que cette clé; ensuite on généralise à l'ensemble des valeurs).
   Ex: `vide/servlet-Delete?table=personne&cle=12`
   Cette fois on génère un formulaire par ligne.

6. Ajoutez à la page Select un lien `Del` en fin de chaque ligne permettant d'appeler cette Servlet pour effacer la ligne correspondante.

7. Créer une servlet `Update.java` L'appel à cette servlet fournira une page HTML présentant les données de la ligne choisie dans des champs `text` inclus dans un formulaire. Ce formulaire appellera, après validation, une autre servlet qui effectuera la mise à jour de la base.
   Ex: `vide/servlet-update?table=personne&cle=12`
   Cette fois on génère des formulaires pré-renseignés. La servlet a besoin des anciennes et des nouvelles valeurs.

8. Ajoutez à la page Select un lien `Mod` en fin de chaque ligne permettant d'appeler cette Servlet pour modifier la ligne correspondante.

A la fin de ce projet vous devez donc avoir une et une seule page visible mais qui permet d'effectuer les 4 opérations de base du DML SQL quelle que soit la table.

Conseils et remarques

- Afficher la requête avant de l'exécuter ! Cela vous aidera grandement à débugguer !
- Les exceptions SQL doivent être "trappées" (mettre un `try-catch`)
- Toujours afficher le message d'erreur (`e.getMessage()`) dans tous les `catch` !
- En phase de mise au point, si c'est possible, passez les formulaires en `GET`. Cela permet de facilement voir dans la servlet appelée si les paramètres sont passés correctement.
- Bannir les chemins absolus; préférez les chemins relatifs (ni `localhost`, ni `8080` ni `vide` dans aucune page; utiliser `grep 8080 *java` par ex)
- Centralisez les accès à la base : soit dans web.xml, soit en utilisant DS.java et un fichier properties.
- Une servlet est un objet. Plus il y a d'objets, mieux c'est. On peut toujours tout mettre dans une seule servlet, comme dans n'importe quel objet, mais la segmentation en outils spécialisés est préférable pour la réutilisation.
- Il n'y a pas systématiquement bijection entre pages visibles et servlets. Pour une seule et même page visible, il faut souvent plusieurs servlets. C'est le cas ici : 1 seule page visible, mais 4 servlets