IHM Qt et base de données

Ce TP a pour but d'améliorer votre IHM développée dans le TP Qt précédent en la liant à une BDD MySQL. Nous vous proposons :

- soit d'utiliser une BDD existante
- soit d'utiliser votre propre BDD en local ou installée sur un serveur distant

Compétences travaillées dans ce TP

- Installation et test d'une nouvelle bibliothèque : SOCI
- Intégration dans un code existant
- Conception objet

1. Installation et test de SOCI

SOCI est une bibliothèque C++ permettant de se connecter à une BDD (MySQL, Oracle, PostGreSQL…) et d'interagir avec elle. A la base, SOCI signifiait Simple Oracle Call Interface, mais désormais, plusieurs moteurs de base de données se sont ajoutés à la bibliothèque.

1. Installation

Si ce n'est pas fait, installez SOCI et le driver mySQL sur WSL :

```
sudo apt install libsoci-dev libmysqlclient-dev
```

ou (sur debian9)

```
sudo apt install libsoci-dev default-libmysqlclient-dev
```

Si cela ne fonctionne pas, vos dépôts ne sont peut-être pas à jour.

2. Test

Créez un nouveau projet de type C++, qui va vous servir à tester SOCI.

Ajoutez les lignes suivantes dans votre CMakeLists avant la ligne add_executable :

```
include_directories(/usr/include/soci)
include_directories(/usr/include/soci/mysql)
include_directories(/usr/include/mysql)
```

Et la ligne suivante après add_executable :

```
target_link_libraries(NOMDEVOTREPROJET soci_core soci_mysql)
```

Remplacez la chaine NODEVOTREPROJET par… le nom de votre projet

Testez SOCI avec le main suivant :

```
#include <iostream>
using namespace std;
#include <exception>
#include <soci.h>
#include <soci-mysql.h>

int main(int argc, char** argv) {
 try {
  soci::session session("mysql://db=inventaire host=vps583945.ovh.net
user='guest' password='isen35'");
  soci::rowset<soci::row> rs = (session.prepare << "SELECT id,nom,quantite
FROM articles");
  for (soci::rowset<soci::row>::const_iterator it = rs.begin(); it !=
rs.end(); ++it) {
    soci::row const& row = *it;
    cout << row.get<int>(0) << " " << row.get<std::string>(1) << " " <<
row.get<int>(2) << endl;
   }
  } catch (const exception &e) {
      cerr << "Error: " << e.what() << endl;
 }
}
```

Vous pouvez bien sûr modifier les options de connexion si vous utilisez une autre BDD (attention à également modifier la requête et les affichages). Si vous voulez consulter une BDD d'un autre type que MySQL, il faudra aussi installer les paquets nécessaires, et modifier les parties include_directories() et target_link_libraries() en conséquence.

Remarque : Vous pourrez observer des avertissements à la compilation du type :

```
warning: template<class> class std::auto_ptr is deprecated [-
Wdeprecated-declarations]
```

Ignorez-les, ils sont déclenchés par du code de la bibliothèque SOCI.

3. Petit bonus avant de continuer

Ouvrir une console bash (WSL) dans CLion

Par défaut, quand on ouvre l'onglet « Terminal » dans CLion (en bas), c'est une invite de commande Windows qui est proposée. On peut changer ce comportement en configurant le terminal, pour par exemple ouvrir une invite de commande Linux. Pour cela, allez dans Settings -> Tools -> Terminal, et changez le champ « Shell path » à « C:\Windows\System32\bash.exe » (c'est cmd.exe qui est lancé de base).

Sous bash, allez dans le répertoire où se trouve l'exécutable et tapez :

```
ldd nondevotreexecutable
```

Vérifiez que SOCI est bien lié à votre exécutable. Quelle est la version de SOCI qui est utilisée ?

2. Une classe dédiée à la BDD

Créez une classe permettant de réaliser :

- la connexion à la BDD
- la récupération des informations dans des objets adaptés

Les choix vous sont laissés libres ici, vos connaissances en C++ vous permettent de faire les bons. N'hésitez pas à en discuter entre vous et à demander à votre enseignant s'ils sont pertinents !

Testez le plus souvent possible votre code afin de vous assurer du bon fonctionnement de votre classe.

Il est important que le processus de communication à la BDD soit séparé dans une classe dédiée (vous auriez peut-être eu envie de tout écrire dans le code de la classe de votre IHM…). Mais c'est une des bases de la programmation graphique : si jamais on décide de changer le mode de stockage des données contenues dans la BDD, il ne doit y avoir qu'un minimum de classe à modifier (si par exemple les données sont mises dans un fichier XML ou CSV, si on change de BDD…). L'idée est de n'avoir absolument rien à modifier du côté de l'IHM. Ce principe de séparation est la base de beaucoup de schémas de conception en développement objet : Modèle-Vue-Contrôleur pour les IHM, DAO (Data Access Objet) pour les bases de données…

3. Intégration dans l'IHM

Si votre classe BDD est bien écrite, vous ne devez avoir que peu de changements à réaliser dans votre classe d'affichage. Faites-en sorte de lier les différents éléments de votre code pour le rendre pleinement opérationnel.

4. Améliorations

Ajoutez les possibilités suivantes :

- Utiliser votre IHM pour envoyer des requêtes à la BDD et afficher le résultat
- Si vous travaillez avec la base "Inventaire", ajoutez le fait de pouvoir ajouter ou retirer un article (en termes de quantité).
- Si vous travaillez avec la base "Inventaire", ajoutez le fait de pouvoir ajouter un article à la base 'Inventaire' (évitez de pouvoir les supprimer afin que la table ne soit jamais vide, vous êtes plusieurs promotions à travailler dessus…). N'oubliez pas de mettre à jour l'affichage sur votre IHM

Ces modifications entraient bien entendu des changements à opérer dans vos différentes classes d'affichage et de connexion à la BDD. Procédez étape par étape, en validant les modifications le plus souvent possible par des tests opérationnels.